

PROGRAMMER'S REFERENCE MANUAL

(Includes CPU32 Instructions)



MOTOROLA

Introduction	1
Integer Instructions	2
Floating-Point Instructions	3
Supervisor (Privileged) Instructions	4
CPU32 Instruction Summary	5
Instruction Format Summary	6
Processor Instruction Summary	A

1 Introduction

2 Integer Instructions

3 Floating-Point Instructions

4 Supervisor (Privileged) Instructions


5 CPU32 Instruction Summary

6 Instruction Format Summary

A Processor Instruction Summary



M68000 FAMILY PROGRAMMER'S REFERENCE MANUAL

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not authorized for use as components in life support devices or systems intended for surgical implant into the body or intended to support or sustain life. Buyer agrees to notify Motorola of any such intended end use whereupon Motorola shall determine availability and suitability of its product or products for the use intended. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Employment Opportunity/Affirmative Action Employer.

PREFACE

This manual contains detailed information about the software instructions used by the microprocessors and coprocessors in the M68000 Family. This manual is organized as follows:

SECTION 1 INTRODUCTION This section contains extensive information on the instruction set (groups by operational category), descriptions of the operands used by instructions, guidelines on how to use the instructions, and details of the conventions and abbreviations used in the instruction descriptions.

SECTION 2 INTEGER INSTRUCTIONS In this section, all integer instruction (nonprivileged) operations are described in detail and organized alphabetically.

SECTION 3 FLOATING-POINT INSTRUCTIONS In this section, all nonprivileged floating-point instructions are detailed.

SECTION 4 SUPERVISOR (PRIVILEGED) INSTRUCTIONS This section contains the operational details for all of the privileged instructions.

SECTION 5 CPU32 INSTRUCTION SUMMARY This section contains information on the CPU32 instructions and addressing modes.

SECTION 6 INSTRUCTION FORMAT SUMMARY This section contains all the M68000 instruction formats (binary), arranged alphabetically.

APPENDIX A PROCESSOR INSTRUCTION SUMMARY This section contains a table listing of all the M68000 Family instructions arranged alphabetically and cross-referenced to show which instructions are applicable to which processors. Each processor is also listed with a table of the instructions and the addressing modes that apply to that processor.

TABLE OF CONTENTS

Paragraph Number	Title	Page Number
	Section 1	
	Introduction	
1.1	Instruction Format	1-1
1.1.1	Data Movement Instructions	1-4
1.1.2	Integer Arithmetic Instructions	1-5
1.1.3	Logical Instructions	1-7
1.1.4	Shift and Rotate Instructions	1-7
1.1.5	Bit Manipulation Instructions	1-8
1.1.6	Bit Field Instructions	1-9
1.1.7	Binary Coded Decimal Instructions	1-10
1.1.8	Program Control Instructions	1-10
1.1.9	System Control Instructions	1-12
1.1.10	Memory Management Unit (MMU) Instructions	1-14
1.1.11	Cache Instructions Instruction (MC68040)	1-14
1.1.12	Multiprocessor Instructions	1-15
1.1.13	Floating-Point Arithmetic Instructions	1-15
1.2	Instruction Set Details	1-17
1.2.1	Notation and Format	1-18
1.2.1.1	Instruction Descriptions	1-20
1.2.1.2	Instructions Examples	1-20
1.2.1.2.1	Using the CAS and CAS2 Instructions	1-20
1.2.1.2.2	Nested Subroutine Calls	1-27
1.2.1.2.3	Bit Field Instructions	1-27
1.2.1.2.4	Pipeline Synchronization with the NOP Instruction ...	1-28
1.2.2	Condition Codes	1-28
1.2.2.1	Condition Code Computation	1-30
1.2.2.1	Conditional Tests	1-31
1.3	Floating-Point Details	1-31
1.3.1	Computational Accuracy	1-32
1.3.2	Conditional Test Definitions	1-34
1.3.3	Operation Tables	1-37
1.3.4	NaNs	1-38
1.3.5	Operation Post Processing	1-38
1.3.5.1	Setting Floating-Point Condition Codes	1-39
1.3.5.2	Underflow, Round, Overflow	1-39

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	Section 2	
	Integer Instructions	
ABCD		2-2
ADD		2-4
ADDA		2-7
ADDI		2-9
ADDQ		2-11
ADDX		2-13
AND		2-15
ANDI		2-18
ANDI to CCR		2-20
ASL, ASR		2-21
Bcc		2-24
BCHG		2-26
BCLR		2-29
BFCHG		2-32
BFCLR		2-34
BFEXTS		2-36
BFETXU		2-38
BFFF0		2-40
BFINS		2-42
BFSET		2-44
BFTST		2-46
BKPT		2-48
BRA		2-50
BSET		2-51
BSR		2-54
BTST		2-56
CALLM		2-59
CAS, CAS2		2-61
CHK		2-64
CHK2		2-66
CLR		2-68
CMP		2-70
CMPA		2-72
CMPI		2-74
CMPM		2-76
CMP		2-77

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	cpBcc	2-79
	cpDBcc	2-80
	cpGEN.....	2-81
	cpScc	2-82
	cpTRAPcc.....	2-84
	DBcc	2-85
	DIVS, DIVU	2-87
	DIVU, DIVUL	2-90
	EOR	2-93
	EORI	2-95
	EORI to CCR.....	2-97
	EXG	2-98
	EXT, EXTB	2-99
	ILLEGAL	2-100
	JMP	2-101
	JSR	2-102
	LEA	2-103
	LINK	2-105
	LSL, LSR.....	2-107
	MOVE	2-110
	MOVEA	2-113
	MOVE from CCR.....	2-115
	MOVE to CCR.....	2-117
	MOVE to SR.....	2-119
	MOVE 16	2-120
	MOVEM	2-123
	MOVEP	2-127
	MOVEQ	2-130
	MULS	2-131
	MULU	2-134
	NBCD	2-137
	NEG	2-139
	NEGX	2-141
	NOP	2-143
	NOT	2-144
	OR	2-146
	ORI	2-149

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	ORI to CCR	2-151
	PACK	2-152
	PEA	2-155
	PVALID	2-156
	ROL, ROR.....	2-159
	ROXL, ROXR	2-162
	RTD	2-165
	RTM	2-166
	RTR	2-167
	RTS	2-168
	SBCD	2-169
	Scc	2-171
	SUB	2-173
	SUBA	2-176
	SUBI	2-178
	SUBQ	2-180
	SUBX	2-182
	SWAP	2-184
	TAS	2-185
	TRAP	2-187
	TRAP _{cc}	2-188
	TRAPV	2-190
	TST	2-191
	UNLK	2-193
	UNPK	2-194

Section 3 Floating-Point Instructions

FABS	3-4
FACOS	3-8
FADD	3-11
FASIN	3-15
FATAN	3-18
FATANH	3-21
FB _{cc}	3-24
FCMP	3-26
FCOS	3-29

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	FCOSH	3-32
	FDBcc	3-35
	FDIV	3-38
	FETOX	3-42
	FGETEXP	3-45
	FGETMAN	3-48
	FINT	3-51
	FINTRZ	3-54
	FLOG10	3-57
	FLOG2	3-60
	FLOGN	3-63
	FLOGNP1	3-66
	FMOD	3-69
	FMOVE	3-72
	FMOVECR	3-82
	FMOVEM	3-84
	FMUL	3-93
	FNEG	3-97
	FNOP	3-100
	FREM	3-102
	FSCALE	3-105
	FScc	3-108
	FSGLDIV	3-110
	FSGLMUL	3-113
	FSIN	3-116
	FSINCOS	3-117
	FSINH	3-123
	FSQRT	3-126
	FSUB	3-130
	FTAN	3-134
	FTANH	3-137
	FTENTOX	3-140
	FTRAPcc	3-143
	FTST	3-145
	FTWOTOX	3-148

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	Section 4	
	Supervisor (Privileged) Instructions	
	ANDI to SR	4-2
	CINV	4-3
	cpRESTORE	4-5
	cpSAVE	4-7
	CPUSH	4-9
	EORI to SR.....	4-11
	FRESTORE	4-12
	FSAVE	4-15
	MOVE from SR.....	4-18
	MOVE to SR.....	4-20
	MOVE USP.....	4-22
	MOVEC	4-23
	MOVES	4-25
	ORI to SR.....	4-28
	PBcc	4-29
	PDBcc	4-31
	PFLUSH (MC68030).....	4-33
	PFLUSH (MC68040).....	4-36
	PFLUSH, PFLUSHA, PFLUSHS (MC68851).....	4-38
	PFLUSHR	4-41
	PLOAD	4-43
	PMOVE (MC68030).....	4-47
	PMOVE (MC68851).....	4-51
	PRESTORE	4-55
	PSAVE	4-57
	PScc	4-59
	PTEST (MC68030)	4-61
	PTST (MC68040)	4-66
	PTEST (MC68851)	4-68
	PTRAPcc	4-72
	RESET	4-74
	RTE	4-75
	STOP	4-77

TABLE OF CONTENTS (Concluded)

Paragraph Number	Title	Page Number
---------------------	-------	----------------

Section 5 CPU32 Instructions

BGND	5-4
LPSTOP	5-5
TBLS, TBLSN.....	5-6
TBLU, TBLUN.....	5-11

Section 6 Instruction Format Summary

6.1	Instruction Format	6-1
6.1.1	Coprocessor ID Field	6-1
6.1.2	Effective Address Field	6-1
6.1.3	Register/Memory Field	6-1
6.1.4	Source Specifier Field	6-1
6.1.5	Destination Register Field.....	6-2
6.1.6	Conditional Predicate Field	6-2
6.1.7	Shift and Rotate Instructions.....	6-2
6.1.7.1	Count Register Field.....	6-2
6.1.7.2	Register Field	6-2
6.1.8	Size Field	6-4
6.1.9	Opmode Field.....	6-4
6.1.10	Address/Data (A/D) Field.....	6-4
6.1.11	Operation Code Map	6-4

Appendix A Processor Instructions Summary

LIST OF ILLUSTRATIONS

Figure Number	Title	Page Number
1-1	Instruction Word General Format	1-2
1-2	Instruction Description Format	1-21
1-3	Linked List Insertion.....	1-22
1-4	Linked List Deletion.....	1-24
1-5	Doubly-Linked List Insertion.....	1-25
1-6	Doubly-Linked List Deletion	1-26
1-7	Operation Table Example (FADD Instruction)	1-37

LIST OF TABLES

Table Number	Title	Page Number
1-1	Data Movement Operations	1-5
1-2	Integer Arithmetic Operations	1-6
1-3	Logical Operations	1-7
1-4	Shift and Rotate Operations	1-8
1-5	Bit Manipulation Operations	1-9
1-6	Bit Field Operations	1-9
1-7	Binary Coded Decimal Operations	1-10
1-8	Program Control Operations	1-11
1-9	FPU Conditional Test Operations	1-12
1-10	System Control Operations	1-13
1-11	MMU Instructions	1-14
1-12	Cache Instructions	1-14
1-13	Multiprocessor Operations	1-15
1-14	Dyadic Floating-Point Operation Format	1-16
1-15	Dyadic Floating-Point Operations	1-16
1-16	Monadic Floating-Point Operation Format	1-17
1-17	Monadic Floating-Point Operations	1-17
1-18	Condition Code Computations	1-30
1-19	Conditional Tests	1-32
1-20	IEEE Nonaware Test	1-36
1-21	IEEE Aware Test	1-36
1-22	Miscellaneous Tests	1-37

SECTION 1

INTRODUCTION

This manual contains detailed information about each of the software instructions used by the microprocessors and coprocessors in the M68000 Family. The M68000 Family includes:

MC68000	— 16-/32-Bit Microprocessor
MC68HC000	— Low Power 16-/32-Bit Microprocessor
MC68008	— 16-Bit Microprocessor with 8-Bit Data Bus
MC68010	— 16-/32-Bit Virtual Memory Microprocessor
MC68020	— 32-Bit Virtual Memory Microprocessor
MC68030	— Second Generation 32-Bit Enhanced Microprocessor
MC68040	— Third Generation 32-Bit Microprocessor
MC68851	— Paged Memory Management Unit
MC68881	— Floating-Point Coprocessor
MC68882	— Enhanced Floating-Point Coprocessor

1.1 INSTRUCTION FORMAT

This section briefly describes the M68000 instruction set in detail using the Motorola assembly language syntax and notation. It includes descriptions of the instruction format and the operands used by instructions, followed by a summary of the instruction set. The integer condition codes and floating-point details are discussed. This main instruction set listing, arranged in alphabetical order, gives detailed descriptions of the operation of each instruction. Programming examples for selected instructions are presented, followed by an instruction set summary arranged by opcode map.

All instructions consist of at least one word; some have as many as 11 words (see Figure 1-1). The first word of the instruction, called the operation word, specifies the length of the instruction and the operation to be performed. The remaining words, called extension words, further specify the instruction and operands. These words may be floating-point command words, conditional predicates, immediate operands, extensions to the effective address mode specified in the operation word, branch displacements, bit number or bit field specifications, special register specifications, trap operands, pack/unpack constants, or argument counts.

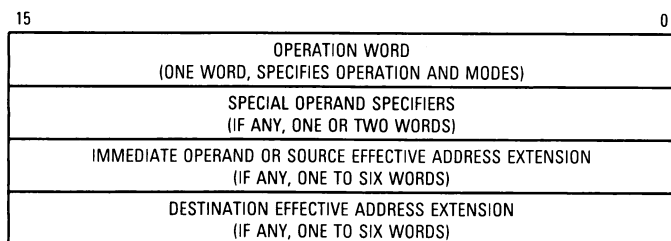


Figure 1-1. Instruction Word General Format

Besides the operation code, which specifies the function to be performed, an instruction defines the location of every operand for the function. Instructions specify an operand location in one of three ways:

- **Register Specification** — A register field of the instruction contains the number of the register.
- **Effective Address** — An effective address field of the instruction contains address mode information.
- **Implicit Reference** — The definition of an instruction implies the use of specific registers.

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

Effective address information includes the registers, displacements, and absolute addresses for the effective address mode.

Certain instructions operate on specific registers. These instructions imply the required registers.

The instructions form a set of tools to perform the following operations:

Data Movement	Binary Coded Decimal Arithmetic
Integer Arithmetic	Program Control
Floating-Point Arithmetic	System Control
Logical	Memory Management
Shift and Rotate	Cache Maintenance
Bit Manipulation	Multiprocessor Communications
Bit Field Manipulation	Dyadic Operation
	Monadic Operation

Each instruction type is described in detail in the following paragraphs.

The following notations are used in this section. In the operand syntax statements of the instruction definitions, the operand on the right is the destination operand.

- An = any address register, A7–A0
- Dn = any data register, D7–D0
- Rn = any address or data register
- CCR = condition code register (lower byte of status register)
 - cc = condition codes from CCR
- ccc = index into the MC68881/MC68882 constant ROM
- SR = status register
- SP = active stack pointer
- USP = user stack pointer
- ISP = supervisor/interrupt stack pointer
- MSP = supervisor/master stack pointer
- SSP = supervisor (master or interrupt) stack pointer
- DFC = destination function code register
- SFC = source function code register
- Rc = control register (VBR, SFC, DFC, CACR SRP, URP, TC, DTT0, DTT1, ITT0, ITT1, MMUSR)
- MMUSR = MMU status register
- B, W, L = specifies a signed integer data type (twos complement) of byte, word, or long word
- S = single precision real data format (32 bits)
- D = double precision real data format (64 bits)
- X = extended precision real data format (96 bits, 16 bits unused)
- P = packed BCD real data format (96 bits, 12 bytes)
- FPm, FPN = any floating-point data register FP7–FP0
- FPcr = floating-point system control register (FPCR, FPSR, or FPIAR)
- k = a twos complement signed integer (–64 to +17) that specifies the format of a number to be stored in the packed decimal format
- d = displacement; d₁₆ is a 16-bit displacement
- <ea> = effective address
- list = list of registers, for example D3–D0
- #<data> = immediate data; a literal integer
- {offset:width} = bit field selection
- label = assemble program label
- [m] = bit m of an operand

[m:n]=bits m through n of operand
 X=extend (X) bit in CCR
 N=negative (N) bit in CCR
 Z=zero (Z) bit in CCR
 V=overflow (V) bit in CCR
 C=carry (C) bit in CCR
 +=arithmetic addition or postincrement indicator
 -=arithmetic subtraction or predecrement indicator
 ×=arithmetic multiplication
 ÷=arithmetic division or conjunction symbol
 ~ =invert; operand is logically complemented
 Λ=logical AND
 V=logical OR
 ⊕=logical exclusive OR
 Dc=data register, D7–D0 used during compare
 Du=data register, D7–D0 used during update
 Dr, Dq=data registers, remainder or quotient of divide
 Dh, Dl=data registers, high or low order 32 bits of product
 MSW=most significant word
 LSW=least significant word
 MSB=most significant bit
 FC=function code
 {R/W}=read or write indicator
 [An]=address extensions

1.1.1 Data Movement Instructions

The MOVE and FMOVE instructions with their associated addressing modes are the basic means of transferring and storing addresses and data. MOVE instructions transfer byte, word, and long word operands from memory to memory, memory to register, register to memory, and register to register. Address movement instructions (MOVE or MOVEA) transfer word and long word operands and ensure that only valid address manipulations are executed. In addition to the general MOVE instructions, there are several special data movement instructions: move 16-byte block (MOVE16), move multiple registers (MOVEM), move peripheral data (MOVEP), move quick (MOVEQ), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), and unlink stack (UNLK). The MOVE16 instruction is an MC68040 extension to the M68000 instruction set.

The FMOVE instructions move operands into, between, and from the floating-point data registers. Data format conversion functions for the FPU instructions are implicitly supported since all external data formats movement of

operands to and from the floating-point control and status registers (FPCR, FPSR, and FPIAR). For operands moved into a floating-point data register, FMOVE and FDMOVE explicitly select single and double precision rounding of the result, respectively. FMOVEM moves any combination of either floating-point data registers or floating-point control registers. Table 1-1 is a summary of the integer and floating-point data movement operations.

Table 1-1. Data Movement Operations

Instruction	Operand Syntax	Operand Size	Operation
EXG	Rn, Rn	32	Rn \leftrightarrow Rn
FMOVE	FPm,FPn <ea>,FPn FPm,<ea> <ea>,FPcr FPcr,<ea>	X B,W,L,S,D,X,P B,W,L,S,D,X,P 32 32	source \leftrightarrow destination
FSMOVE, FDMOVE	FPm,FPn <ea>,FPn	X B,W,L,S,D,X	source \leftrightarrow destination, round destination to single or double precision
FMOVEM	<ea>,<list> ¹ <ea>,Dn <list> ¹ ,<ea> Dn,<ea>	32,X X 32,X X	listed registers \leftrightarrow destination source \leftrightarrow listed registers
LEA	<ea>,An	32	<ea> \leftarrow An
LINK	An,<d>	16,32	Sp - 4 \leftarrow SP; An \leftarrow (SP); SP \leftarrow An, SP + D \leftarrow SP
MOVE MOVE16 MOVEA	<ea>,<ea> <ea>,<ea> <ea>,An	8,16,32 16 bytes 16,32 \leftrightarrow 32	source \leftrightarrow destination aligned 16-byte block \leftrightarrow destination
MOVEM	list,<ea> <ea>,list	16,32 16,32 \leftrightarrow 32	listed registers \leftrightarrow destination source \leftrightarrow listed registers
MOVEP	Dn, (d ₁₆ ,An) (d ₁₆ ,An),Dn	16,32	Dn[31:24] \leftrightarrow (An + d); Dn[23:16] \leftrightarrow An + d + 2; Dn[15:8] \leftrightarrow (An + d + 4); Dn[7:0] \leftrightarrow (An + d + 6) (An + d) \leftrightarrow Dn[31:24]; (An + d + 2) \leftrightarrow Dn[23:16]; (An + d + 4) \leftrightarrow Dn[15:8]; (An + d + 6) \leftrightarrow Dn[7:0]
MOVEQ	#<data>,Dn	8 \leftrightarrow 32	immediate data \leftrightarrow destination
PEA	<ea>	32	SP - 4 \leftarrow SP; <ea> \leftarrow (SP)
UNLK	An	32	An \leftarrow SP; (SP) \leftarrow An; SP + 4 \leftarrow SP

NOTE 1: The register list may include any combination of the eight floating-point data registers, or it may contain any combination of the three control registers (FPCR, FPSR, and FPIAR). If the register list mask resides in a data register, only floating-point data registers may be specified.

1.1.2 Integer Arithmetic Instructions

The integer arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP, CMPM, CMP2), clear (CLR), and negate (NEG). The instruction set includes ADD, CMP, and SUB instructions for both address and data operations with all operand sizes valid for data operations. Address operands consist of 16 or 32 bits. The clear and negate instructions apply to all sizes of data operands.

Signed and unsigned MUL and DIV instructions include:

- Word multiply to produce a long word product
- Long word multiply to produce and long word or quad word product
- Division of a long word divided by a word divisor (word quotient and word remainder)
- Division of a long word or quad word dividend by a long word divisor (long word quotient and long word remainder)

A set of extended instructions provides multiprecision and mixed size arithmetic. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX). Refer to Table 1-2 for a summary of the integer arithmetic operations.

Table 1-2. Integer Arithmetic Operations

Instruction	Operand Syntax	Operand Size	Operation
ADD	Dn,(ea)	8, 16, 32	source + destination ∇ destination
ADDA	(ea),Dn (ea),An	8, 16, 32 16, 32	
ADDI	#{data),(ea)	8, 16, 32	immediate data + destination ∇ destination
ADDQ	#{data),(ea)	8, 16, 32	
ADDX	Dn,Dn -(An), -(An)	8, 16, 32 8, 16, 32	source + destination + X ∇ destination
CLR	(ea)	8, 16, 32	0 ∇ destination
CMP	(ea),Dn	8, 16, 32	destination - source
CMPA	(ea),An	16, 32	
CMPI	#{data),(ea)	8, 16, 32	destination - immediate data
CMPM	(An) + ,(An) +	8, 16, 32	destination - source
CMP2	(ea),Rn	8, 16, 32	lower bound < = Rn < = upper bound
DIVS/DIVU	(ea),Dn (ea),Dr:Dq	32/16 ∇ 16:16 64/32 ∇ 32:32	destination/source ∇ destination (signed or unsigned)
DIVSL/DIVUL	(ea),Dq (ea),Dr:Dq	32/32 ∇ 32 32/32 ∇ 32:32	
EXT	Dn	8 ∇ 16	sign extended destination ∇ destination
EXTB	Dn	16 ∇ 32 8 ∇ 32	
MULS/MULU	(ea),Dn (ea),Dl (ea),Dh:Dl	16 \times 16 ∇ 32 32 \times 32 ∇ 32 32 \times 32 ∇ 64	source \times destination ∇ destination (signed or unsigned)
NEG	(ea)	8, 16, 32	0 - destination ∇ destination
NEGX	(ea)	8, 16, 32	0 - destination - X ∇ destination
SUB	(ea),Dn	8, 16, 32	destination = source ∇ destination
SUBA	Dn,(ea) (ea),An	8, 16, 32 16, 32	
SUBI	#{data),(ea)	8, 16, 32	destination - immediate data ∇ destination
SUBQ	#{data),(ea)	8, 16, 32	
SUBX	Dn,Dn -(An), -(An)	8, 16, 32 8, 16, 32	destination - source - X ∇ destination

1.1.3 Logical Instructions

The logical operation instructions (AND, OR, EOR, and NOT) perform logical operations with all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 1-3 summarizes the logical operations.

Table 1-3. Logical Operations

Instruction	Operand Syntax	Operand Size	Operation
AND	(ea),Dn Dn,(ea)	8, 16, 32 8, 16, 32	source \wedge destination \nrightarrow destination
ANDI	#<data>,<ea>	8, 16, 32	immediate data \wedge destination \nrightarrow destination
EOR	Dn,<data>,<ea>	8, 16, 32	source \oplus destination \nrightarrow destination
EORI	#(data),(ea)	8, 16, 32	immediate data \oplus destination \nrightarrow destination
NOT	(ea)	8, 16, 32	\sim destination \nrightarrow destination
OR	(ea),Dn Dn,(ea)	8, 16, 32 8, 16, 32	source \vee destination \nrightarrow destination
ORI	#(data),(ea)	8, 16, 32	immediate data \vee destination \nrightarrow destination

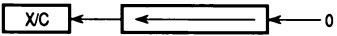
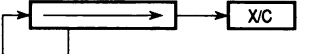
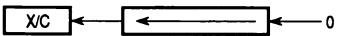
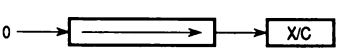
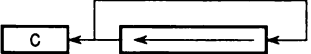
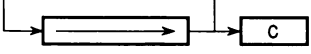
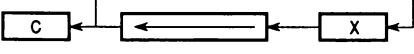
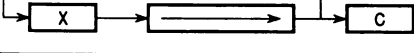
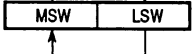
1.1.4 Shift and Rotate Instructions

The arithmetic shift instructions (ASR and ASL) and logical shift instructions (LSR and LSL) provide shift operations in both directions. The ROR, ROL, ROXR, and ROXL instructions perform rotate (circular shift) operations, with and without the extend bit. All shift and rotate operations can be performed on either registers or memory.

Register shift and rotate operations shift all operand sizes. The shift count may be specified in the instruction operation word (to shift from 1–8 places) or in a register (modulo 64 shift count).

Memory shift and rotate operations shift word-length operands one bit position only. The SWAP instruction exchanges the 16-bit halves of a register. Performance of shift/rotate instructions is enhanced so that use of the ROR and ROL instructions with a shift count of eight allows fast byte swapping. Table 1-4 is a summary of the shift and rotate operations.

Table 1-4. Shift and Rotate Operations

Instruction	Operand Syntax	Operand Size	Operation
ASL	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
ASR	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
LSL	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
LSR	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
ROL	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
ROR	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
ROXL	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
ROXR	Dn,Dn #(data),Dn (ea)	8, 16, 32 8, 16, 32 16	
SWAP	Dn	32	

1.1.5 Bit Manipulation Instructions

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). All bit manipulation operations can be performed on either registers or memory. The bit number is specified as immediate data or in a data register. Register operands are 32 bits long, and memory operands are 8 bits long. In Table 1-5, the summary of the bit manipulation operations, Z refers to the zero bit of the status register.

Table 1-5. Bit Manipulation Operations

Instruction	Operand Syntax	Operand Size	Operation
BCHG	Dn,(ea) #(data),(ea)	8, 32 8, 32	~ ((bit number) of destination) ♦ Z ♦ bit of destination
BCLR	Dn,(ea) #(data),(ea)	8, 32 8, 32	~ ((bit number) of destination) ♦ Z; 0 ♦ bit of destination
BSET	Dn,(ea) #(data),(ea)	8, 32 8, 32	~ ((bit number) of destination) ♦ Z; 1 ♦ bit of destination
BTST	Dn,(ea) #(data),(ea)	8, 32 8, 32	~ ((bit number) of destination) ♦ Z

1.1.6 Bit Field Instructions

The M68000 architecture supports variable length bit field operations on fields of up to 32 bits. The bit field insert (BFINS) instruction inserts a value into a bit field. Bit field extract unsigned (BFEXTU) and bit field extract signed (BFEXTS) extract a value from the field. Bit field find first one (BFFFO) finds the first bit that is set in a bit field. Also included are instructions that are analogous to the bit manipulation operations; bit field test (BFTST), bit field test and set (BFSET), bit field test and clear (BFCLR), and bit field test and change (BFCHG). Table 1-6 is a summary of the bit field operations.

Table 1-6. Bit Field Operations

Instruction	Operand Syntax	Operand Size	Operation
BFCHG	(ea) {offset:width}	1–32	~ Field ♦ Field
BFCLR	(ea) {offset:width}	1–32	0's ♦ Field
BFEXTS	(ea) {offset:width},Dn	1–32	Field ♦ Dn; Sign Extended
BFEXTU	(ea) {offset:width},Dn	1–32	Field ♦ Dn; Zero Extended
BFFFO	(ea) {offset:width},Dn	1–32	Scan for first bit set in field; offset ♦ Dn
BFINS	Dn,(ea) {offset:width}	1–32	Dn ♦ Field
BFSET	(ea) {offset:width}	1–32	1's ♦ Field
BFTST	(ea) {offset:width}	1–32	Field MSB ♦ N; ~ (OR of all bits in field) ♦ Z

NOTE: All bit field instructions set the N and Z bits as shown for BFTST before performing the specified operation.

1.1.7 Binary Coded Decimal Instructions

Five instructions support operations on binary coded decimal (BCD) numbers. The arithmetic operations on packed binary coded decimal numbers are: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). PACK and UNPACK instructions aid in the conversion of byte encoded numeric data, such as ASCII or EBCDIC strings, to BCD data and vice versa. Table 1-7 is a summary of the binary coded decimal operations.

Table 1-7. Binary Coded Decimal Operations

Instruction	Operand Syntax	Operand Size	Operation
ABCD	Dn,Dn – (An), – (An)	8 8	source ₁₀ + destination ₁₀ + X ∇ destination
NBCD	(ea)	8	0 – destination ₁₀ – X ∇ destination
PACK	– (An), – (An) #(data) Dn,Dn,#(data)	16 ∇ 8 16 ∇ 8	unpacked source + immediate data ∇ packed destination
SBCD	Dn,Dn – (An), – (An)	8 8	destination ₁₀ – source ₁₀ – X ∇ destination
UNPK	– (An), – (An) #(data) Dn,Dn,#(data)	8 ∇ 16 8 ∇ 16	packed source ∇ unpacked source unpacked source + immediate data ∇ unpacked destination

1.1.8 Program Control Instructions

A set of subroutine call and return instructions and conditional and unconditional branch instructions perform program control operations. Also included are test operand instructions (TST and FTST) that set the integer or floating-point condition codes for use by the other program and system control instructions, and a no operation instruction (NOP) that may be used to force synchronization of the internal pipelines. Table 1-8 summarizes these instructions.

The conditional mnemonics for the floating-point conditional instructions are shown in Table 1-9, along with the conditional test function. The FPU supports 32 conditional tests that are separated into two groups; 16 that cause an exception if an unordered condition is present when the conditional test is attempted, and 16 that do not cause an exception if an unordered condition is present. (An unordered condition occurs when an input to an arithmetic operation is a NAN.) Refer to **1.3.2 Conditional Test Definitions** for a detailed description of the conditional equation used by each test.

Table 1-8. Program Control Operations

Instruction	Operand Syntax	Operand Size	Operation
Integer and Floating-Point Conditional			
Bcc, FBcc	<label>	8,16,32	if condition true, then PC + d ∇ PC
DBcc, FDBcc	Dn,<label>	16	if condition false, then Dn - 1 ∇ Dn if Dn \neq -1, then PC + d ∇ PC
Scc, FScc	<ea>	8	if condition true, then 1's ∇ destination; else 0's ∇ destination
Unconditional			
BRA	<label>	8,16,32	PC + d ∇ PC
BSR	<label>	8,16,32	SP - 4 ∇ SP; PC ∇ (SP); PC + d ∇ PC
JMP	<ea>	none	destination ∇ PC
JSR	<ea>	none	SP - 4 ∇ SP; PC ∇ (SP); destination ∇ PC
NOP	none	none	PC + 2 ∇ PC
FNOP	none	none	PC + 4 ∇ PC
Returns			
RTD	#<d>	16	(SP) ∇ PC; SP + 4 + d ∇ SP
RTR	none	none	(SP) ∇ CCR; SP + 2 ∇ SP; (SP) ∇ PC; SP + 4 ∇ SP
RTS	none	none	(SP) ∇ PC; SP + 4 ∇ SP
Test Operand			
TST	<ea>	8, 16, 32	set integer condition codes
FTST	<ea> FPn	B,W,L,S,D,X,P X	set floating-point condition codes

Letters cc in the integer instruction mnemonics Bcc, DBcc, and Scc specify testing one of the following conditions:

CC — Carry clear	GE — Greater or equal
LS — Lower or same	PL — Plus
CS — Carry set	GT — Greater than
LT — Less than	T — Always true*
EQ — Equal	HI — Higher
MI — Minus	VC — Overflow clear
F — Never true*	LE — Less or equal
NE — Not equal	VS — Overflow set

*Not applicable to the Bcc instructions.

Table 1-9. FPU Conditional Test Mnemonics

Exception on Unordered		No Exception on Unordered	
GE	Greater Than or Equal	OGE	Ordered Greater Than or Equal
GL	Greater Than or Less Than	OGI	Ordered Greater Than or Less Than
GLE	Greater Than, Less Than, or Equal	OR	Ordered
GT	Greater Than	OGT	Ordered Greater Than
LE	Less Than or Equal	OLE	Ordered Less Than or Equal
LT	Less Than	OLT	Ordered Less Than
NGE	Not (greater than or equal)	UGE	Unordered or Greater Than Equal
NGL	Not (greater than or less than)	UEQ	Unordered or Equal
NGLE	Not (greater than or less than or equal)	UN	Unordered
NGT	Not Greater Than	UGT	Unordered or Greater Than
NLE	Not (less than or equal)	ULE	Unordered or Less Than or Equal
NLT	Not Less Than	ULT	Unordered or Less Than
SEQ	Signaling Equal	EQ	Equal
SNE	Signaling Not Equal	NE	Not Equal
SF	Signaling Always False	F	Always False
ST	Signaling Always True	T	Always True

1.1.9 System Control Instructions

Privileged instructions, trapping instructions, and instructions that use or modify the condition code register (CCR) provide system control operations. Table 1-10 summarizes these instructions. FSAVE and FRESTORE save and restore the nonuser visible portion of the FPU during context switches in a virtual memory or multitasking system. The conditional trap instructions use the same conditional tests as their corresponding program control instructions and allow an optional 16- or 32-bit immediate operand to be included as part of the instruction for passing parameters to the operating system. All of these instructions cause the processor to flush the instruction pipe. See **1.3.5 Operation Post Processing** for more details on condition codes.

Table 1-10. System Control Operations

Instruction	Operand Syntax	Operand Size	Operation
Privileged			
ANDI	#<data>,SR	16	immediate data \wedge SR \blacklozenge SR
EORI	#<data>,SR	16	immediate data \oplus SR \blacklozenge SR
FRESTORE	<ea>	none	state frame \blacklozenge internal floating-point registers
FSAVE	<ea>	none	internal floating-point registers \blacklozenge state frame
MOVE	<ea>,SR	16	source \blacklozenge SR
	SR,<ea>	16	SR \blacklozenge destination
MOVE	USP,An	32	USP \blacklozenge An
	An,USP	32	An \blacklozenge USP
MOVEC	Rc,Rn	32	Rc \blacklozenge Rn
	Rn,Rc	32	Rn \blacklozenge Rc
MOVES	Rn,<ea>	8,16,32	Rn \blacklozenge destination using DFC
	<ea>,Rn		source using SFC \blacklozenge Rn
ORI	#<data>,SR	16	immediate data \vee SR \blacklozenge SR
RESET	none	none	assert $\overline{\text{RSTO}}$ line
RTE	none	none	(SP) \blacklozenge SR; SP + 2 \blacklozenge SP; (SP) \blacklozenge PC; SP + 4 \blacklozenge SP; Restore stack according to format
STOP	#<data>	16	immediate data \blacklozenge SR; STOP
Trap Generating			
BKPT	#<data>	none	run breakpoint cycle, then trap as illegal instruction
CHK	<ea>,Dn	16,32	if Dn<0 or Dn>(ea), then CHK exception
CHK2	<ea>,Rn	8,16,32	if Rn<lower bound or Rn>upper bound, the CHK exception
ILLEGAL	none	none	SSP - 2 \blacklozenge SSP; Vector Offset \blacklozenge (SSP); SSP - 4 \blacklozenge SSP; PC \blacklozenge (SSP); SSP - 2 \blacklozenge SSP; SR \blacklozenge (SSP); Illegal Instruction Vector Address \blacklozenge PC
TRAP	#<data>	none	SSP - 2 \blacklozenge SSP; Format and Vector Offset \blacklozenge (SSP) SSP - 4 \blacklozenge SSP; PC \blacklozenge (SSP); SSP - 2 \blacklozenge SSP; SR \blacklozenge (SSP); Vector Address \blacklozenge PC
TRAPcc	none #<data>	none 16,32	if cc true, then TRAP exception
FTRAPcc	none #<data>	none 16,32	if floating-point cc true, then TRAP exception
TRAPV	none	none	if V then take overflow TRAP exception
Condition Code Register			
ANDI	#<data>,CCR	8	immediate data \wedge CCR \blacklozenge CCR
EORI	#<data>,CCR	8	immediate data \oplus CCR \blacklozenge CCR
MOVE	<ea>,CCR	16	source \blacklozenge CCR
	CCR,<ea>	16	CCR \blacklozenge destination
ORI	#<data>,CCR	8	immediate data \vee CCR \blacklozenge CCR

1.1.10 Memory Management Unit (MMU) Instructions

The PFLUSH instructions flush the address translation caches (ATCs), and can optionally select only nonglobal entries for flushing. PTEST performs a search of the address translation tables, storing results in the MMU status register and loading the entry into the ATC. Table 1-11 summarizes these instructions.

Table 1-11. MMU Instructions

Instruction	Operand Syntax	Operand Size	Operation
PFLUSHA	none	none	Invalidate all ATC entries
PFLUSHA.N	none	none	Invalidate all nonglobal ATC entries
PFLUSH	(An)	none	Invalidate ATC entries at effective address
PFLUSH.N	(An)	none	Invalidate nonglobal ATC entries at effective address
PTEST	(An)	none	Information about logical address ♦ MMU status register

1.1.11 Cache Control Instructions (MC68040)

The cache instructions provide maintenance functions for managing the instruction and data caches. CINV invalidates cache entries in both caches, and CPUSH pushes dirty data from the data cache to update memory. Both instructions can operate on either or both caches, and can select a single cache line, all lines in a page, or the entire cache. Table 1-12 summarizes these instructions.

Table 1-12. Cache Instructions

Instruction	Operand Syntax	Operand Size	Operation
CINVL	caches,(An)	none	Invalidate cache line
CINVP	caches, (An)	none	Invalidate cache page
CINVA	caches	none	Invalidate entire cache
CPUSHL	caches,(An)	none	Push selected dirty data cache lines, then invalidate
CPUSHP	caches, (An)	none	selected cache lines
CPUSHA	caches	none	

1.1.12 Multiprocessor Instructions

The TAS, CAS, and CAS2 instructions coordinate the operations of processors in multiprocessing systems. These instructions use read-modify-write bus cycles to ensure uninterrupted updating of memory. Coprocessor instructions control the coprocessor operations. Table 1-13 summarizes these instructions.

Table 1-13. Multiprocessor Operations

Instruction	Operand Syntax	Operand Size	Operation
READ-MODIFY-WRITE			
CAS	Dc,Du,(ea)	8, 16, 32	destination ← Dc ◊ CC; if Z then Du ◊ destination else destination ◊ Dc
CAS2	Dc1:Dc2, Du1:Du2, (Rn):(Rn)	16, 32	dual operand CAS
TAS	(ea)	8	destination ← 0; set condition codes; 1 ◊ destination [7]
COPROCESSOR			
cpBcc	(label)	16, 32	if cpcc true then PC + d ◊ PC
cpDBcc	(label),Dn	16	if cpcc false then Dn ← 1 ◊ Dn if Dn ≠ −1, then PC + d ◊ PC
cpGEN	User Defined	User Defined	operand ◊ coprocessor
cpRESTORE	(ea)	none	restore coprocessor state from (ea)
cpSAVE	(ea)	none	save coprocessor state at (ea)
cpScc	(ea)	8	if cpcc true, then 1's ◊ destination; else 0's ◊ destination
cpTRAPcc	none #(data)	none 16, 32	if cpcc true then TRAPcc exception

1.1.13 Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions supported by the MC68040 are an enhanced subset of the MC68881/MC68882 floating-point coprocessor instructions. Several instructions in the MC68040 include explicit single and double precision rounding of the result as part of their operation. For example, the FADD instruction uses the default rounding precision selected in the FPU, while FSADD and FDADD force rounding of the result to single and double precision, respectively. The following paragraphs describe the floating-point instructions, which are organized into two categories of operation: dyadic (requiring two operands), and monadic (requiring one operand).

The dyadic floating-point instructions provide several arithmetic functions that require two input operands, such as add and subtract. For these operations, the first operand may be located in memory, in an integer data register, or in a floating-point data register, and the second operand is always

contained in a floating-point data register. The results of the operation are stored in the register specified as the second operand. All operations support any data format, and return results rounded to either extended precision, or single or double precision for the instructions which explicitly specify the rounding precision (such as FSADD and FDADD). The general format of the dyadic instructions is given in Table 1-14; the available operations are listed in Table 1-15.

Table 1-14. Dyadic Floating-Point Operation Format

Instruction	Operand Syntax	Operand Format	Operation
F(dop)	(ea),FPn FPm,FPn	B,W,L,S,D,X,P X	FPn (function) source ♦ FPn

where:

<dop> is any one of the dyadic operation specifiers.

Table 1-15. Dyadic Floating-Point Operations

Instruction	Function
FADD, FSADD, FDADD	Add
FCMP	Compare
FDIV, FSDIV, FDDIV	Divide
FMOD	Modulo Remainder
FMUL, FSMUL, FDMUL	Multiply
FREM	IEEE Remainder
FSCALE	Scale Exponent
FSUB, FSSUB, FDSUB	Subtract
FSGLDIV, FSGLMUL	Single Precision-Multiply, Divide

The monadic floating-point instructions provide several arithmetic functions that require only one input operand. Unlike the integer counterparts to these functions (e.g., NEG <ea>), a source and a destination may be specified. The operation is performed on the source operand and the result is stored in the destination, which is always a floating-point data register. When the source is not a floating-point data register, all data formats are supported;

the data format is always extended precision for register-to-register operations. The general format of these instructions is shown in Table 1-16, and the available operations are listed in Table 1-17.

Table 1-16. Monadic Floating-Point Operation Format

Instruction	Operand Syntax	Operand Format	Operation
F(mop)	(ea),FPn FPm,FPn FPn	B,W,L,S,D,X,P X X	source ♦ function ♦ FPn FPn ♦ function ♦ FPn

where:

<mop> is any one of the monadic operation specifiers.

Table 1-17. Monadic Floating-Point Operations

Instruction	Function	Instruction	Function
FABS	Absolute Value	FLOGN	$\ln(x)$
FACOS	Arc Cosine	FLOGNP1	$\ln(x + 1)$
FASIN	Arc Sine	FLOG10	$\log_{10}(x)$
FATAN	Hyperbolic Art Tangent	FLOG2	$\log_2(x)$
FCOS	Cosine	FNEG	Negate
FCOSH	Hyperbolic Cosine	FSIN	Sine
FETOX	e^x	FSINH	Hyperbolic Sine
FETOM1	$e^x - 1$	FSQRT	Square Root
FGETEXP	Extract Exponent	FTAN	Tangent
FGETMAN	Extract Mantissa	FTANH	Hyperbolic Tangent
FINT	Extract Integer Part	FTENTOX	10^x
FINTRZ	Extract Integer Part, Rounded-To-Zero	FTWOTOX	2^x

1.2 INSTRUCTION SET DETAILS

Sections 2, 3, 4, and 5 contain detailed information about each instruction in the M68000 Family instruction set. First a description of the notation and format of the instruction description is presented here. The instruction descriptions are arranged in alphabetical order by instruction mnemonic.

1.2.1 Notation and Format

The instruction descriptions use notational conventions for the operands, the subfields and qualifiers, and the operations performed by the instructions. In the syntax descriptions, the left operand is the source operand, and the right operand is the destination operand. The following lists contain the notations used in the instruction descriptions.

Notation for operands:

- PC—Program counter
- SR—Status register
- V—Overflow condition code
- Immediate Data—Immediate data from the instruction
- Source—Source contents
- Destination—Destination contents
- Vector—Location of exception vector
- + inf—Positive infinity
- inf—Negative infinity
- <fmt>—Operand data format: byte (B), word (W), long (L), single (S), double (D), extended (X), or packed (P).
- FPm—One of eight floating-point data registers (always specifies the source register)
- FPn—One of eight floating-point data registers (always specifies the destination register)

Notation for subfields and qualifiers:

- <bit> of <operand>—Selects a single bit of the operand
- <ea>{offset:width}—Selects a bit field
- (<operand>)—The contents of the referenced location
- <operand>10—The operand is binary coded decimal, operations are performed in decimal
- (<address register>)—The register indirect operator
- (<address register>)—Indicates that the operand register points to the memory
- (<address register>)+—Location of the instruction operand — the optional mode qualifiers are –, +, (d), and (d,ix)
- #xxx or #<data>—Immediate data that follows the instruction word(s)

Notations for operations that have two operands, written <operand> <op> <operand>, where <op> is one of the following:

- ↔—The source operand is moved to the destination operand
- ↔—The two operands are exchanged
- ++—The operands are added
- The destination operand is subtracted from the source operand
- ×—The operands are multiplied
- ÷—The source operand is divided by the destination operand
- <—Relational test, true if source operand is less than destination operand
- >—Relational test, true if source operand is greater than destination operand
- V—Logical OR
- ⊕—Logical exclusive OR
- Λ—Logical AND

shifted by, rotated by—The source operand is shifted or rotated by the number of positions specified by the second operand

Notation for single-operand operations:

- ~<operand>—The operand is logically complemented
- <operand>sign-extended—The operand is sign extended, all bits of the upper portion are made equal to the high-order bit of the lower portion
- <operand>tested—The operand is compared to zero and the condition codes are set appropriately

Notation for other operations:

- TRAP—Equivalent to Format/Offset Word ↗ (SSP); SSP – 2 ↗ SSP; PC ↗ (SSP); SSP – 4 ↗ SSP; SR ↗ (SSP); SSP – 2 ↗ SSP; (vector) ↗ PC
- STOP—Enter the stopped state, waiting for interrupts
- If <condition> then—The condition is tested. If true, the operations <operations> else after “then” are performed. If the condition is false and the optional “else” clause is present, the operations after “else” are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.

1.2.1.1 INSTRUCTION DESCRIPTIONS. Figure 1-2 shows the format of the instruction descriptions. The attributes line specifies the size of the operands of an instruction. When an instruction can use operands of more than one size, a suffix is used with the mnemonic of the instruction:

- .B — Byte operands
- .W — Word operands
- .L — Long-word operands
- .S — Single-precision real operands
- .D — Double-precision real operands
- .X — Extended-precision real operands
- .P — Packed BCD real operands

1.2.1.2 INSTRUCTION EXAMPLES. The following paragraphs provide examples of how to use selected instructions.

1.2.1.2.1 Using CAS and CAS2. The CAS instruction compares the value in a memory location with the value in a data register, and copies a second data register into the memory location if the compared values are equal. This provides a means of updating system counters, history information, and globally shared pointers. The instruction uses an indivisible read-modify-write cycle; after CAS reads the memory location; no other instruction can change that location before CAS has written the new value. This provides security in single-processor systems, in multitasking environments, and in multiprocessor environments. In a single-processor system, the operation is protected from instructions of an interrupt routine. In a multitasking environment, no other task can interfere with writing the new value of a system variable. In a multiprocessor environment, the other processors must wait until the CAS instruction completes before accessing a global pointer.

The following code fragment shows a routine to maintain a count, in location SYS-CNTR, of the executions of an operation that may be performed by any process or processor in a system. The routine obtains the current value of the count in register D0 and stores the new count value in register D1. The CAS instruction copies the new count into SYS-CNTR if it is valid. But if another user has incremented the counter between the time the count was stored and the read-modify-write cycle of the CAS instruction, the write por-

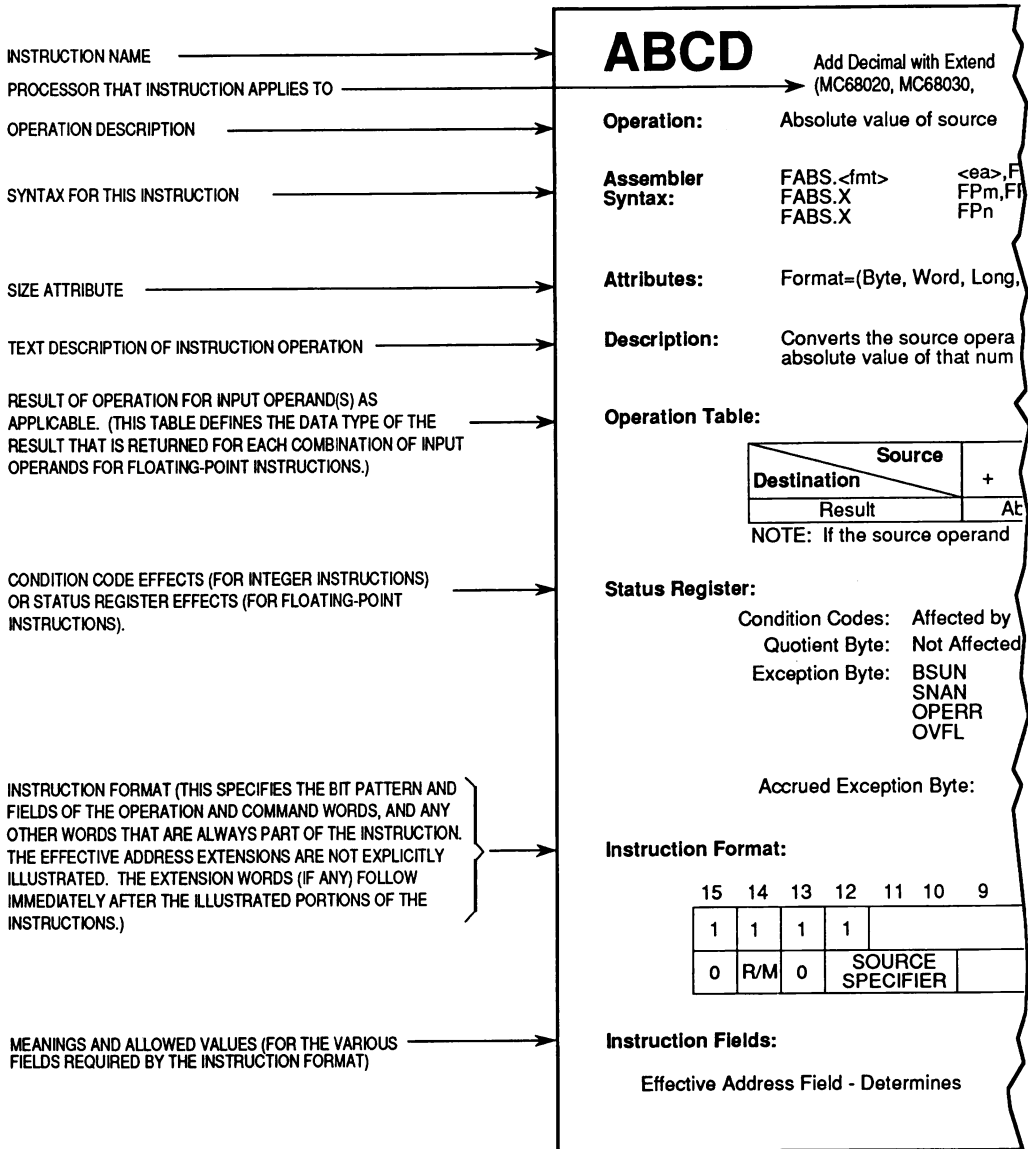


Figure 1-2. Instruction Description Format

tion of the cycle copies the new count in SYS_CNTR into D0, and the routine branches to repeat the test. The following code sequence guarantees that SYS_CNTR is correctly incremented.

	MOVE.W	SYS_CNTR,D0	get the old value of the counter
INC_LOOP	MOVE.W	D0,D1	make a copy of it
	ADDQ.W	#1,D1	and increment it
	CAS.W	D0,D1,SYS_CNTR	if counter value is still the same, update it
	BNE	INC_LOOP	if not, try again

The CAS and CAS2 instructions together allow safe operations in the manipulation of system linked lists. Controlling a single location, HEAD in the example, manages a last-in-first-out linked list (see Figure 1-3). If the list is empty, HEAD contains the NULL pointer (0); otherwise, HEAD contains the address of the element most recently added to the list. The code fragment, shown in Figure 1-3, illustrates the code for inserting an element. The MOVE instructions load the address in location HEAD into D0 and into the NEXT pointer in the element being inserted, and the address of the new element into D1. The CAS instruction stores the address of the inserted element into

SINSERT			ALLOCATE NEW ENTRY, ADDRESS IN A1
	MOVE.L	HEAD,D0	MOVE HEAD POINTER VALUE TO D0
SILoop	MOVE.L	D0,(NEXT,A1)	ESTABLISH FORWARD LINK IN NEW ENTRY
	MOVE.L	A1,D1	MOVE NEW ENTRY POINTER VALUE TO D1
	CAS.L	D0,D1,HEAD	IF WE STILL POINT TO TOP OF STACK, UPDATE THE HEAD POINTER
	BNE	SILoop	IF NOT, TRY AGAIN

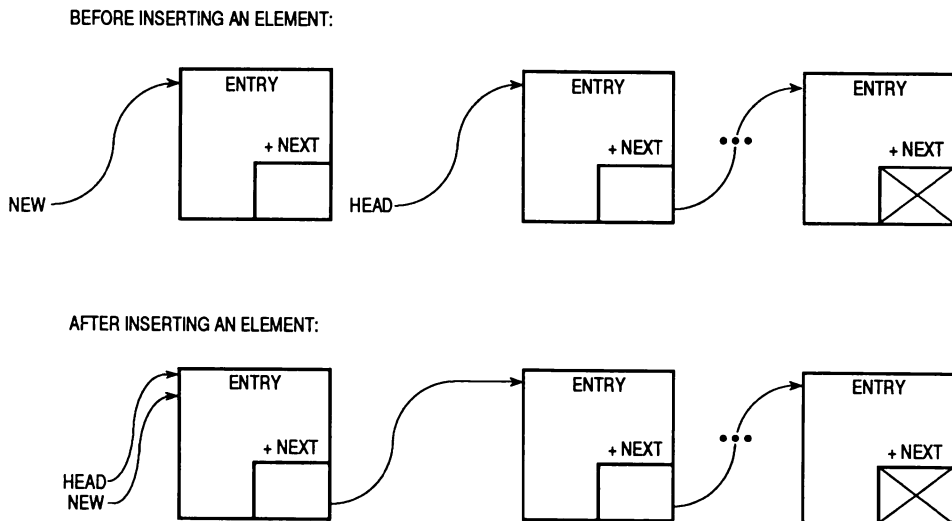


Figure 1-3. Linked List Insertion

location HEAD if the address in HEAD remains unaltered. If HEAD contains a new address, the instruction loads the new address into D0 and branches to the second MOVE instruction to try again.

The CAS2 instruction is similar to the CAS instruction except that it performs two comparisons and updates two variables when the results of the comparisons are equal. If the results of both comparisons are equal, CAS2 copies new values into the destination addresses. If the result of either comparison is not equal, the instruction copies the values in the destination addresses into the compare operands.

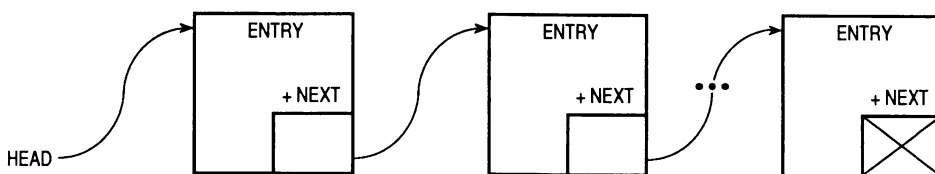
The next code (see Figure 1-4) fragment shows the use of a CAS2 instruction to delete an element from a linked list. The first LEA instruction loads the effective address of HEAD into A0. The MOVE instruction loads the address in pointer HEAD into D0. The TST instruction checks for an empty list, and the BEQ instruction branches to a routine at label SDEMPTY if the list is empty. Otherwise, a second LEA instruction loads the address of the NEXT pointer in the newest element on the list into A1, and the following MOVE instruction loads the pointer contents into D1. The CAS2 instruction compares the address of the newest structure to the value in HEAD and the address in D1 to the pointer in the address in A1. If no element has been inserted or deleted by another routine while this routine has been executing, the results of these comparisons are equal, and the CAS2 instruction stores the new value into location HEAD. If an element has been inserted or deleted, the CAS2 instruction loads the new address in location HEAD into D0, and the BNE instruction branches to the TST instruction to try again.

The CAS2 instruction can also be used to correctly maintain a first-in first-out doubly-linked list. A doubly-linked list needs two controlled locations, LIST_PUT and LIST_GET, which contain pointers to the last element inserted in the list and the next to be removed, respectively. If the list is empty, both pointers are NULL (0).

The code fragment in Figure 1-5 illustrates the insertion of an element in a doubly-linked list. The first two instructions load the effective addresses of LIST_PUT and LIST_GET into registers A0 and A1, respectively. The next instruction moves the address of the new element into register D2. Another MOVE instruction moves the address in LIST_PUT into register D0. At label DILOOP, a TST instruction tests the value in D0, and the BEQ instruction branches to the MOVE instruction when D0 is equal to zero. Assuming the list is empty, this MOVE instruction is executed next; it moves the zero in D0 into the NEXT and LAST pointers of the new element. Then the CAS2 instruction moves the address of the new element into both LIST_PUT and

SDELETE	LEA	HEAD, A0	LOAD ADDRESS OF HEAD POINTER INTO A0
	MOVE.L	(A0), D0	MOVE VALUE OF HEAD POINTER INTO D0
SDLOOP	TST.L	D0	CHECK FOR NULL HEAD POINTER
	BEQ	SDEEMPTY	IF EMPTY, NOTHING TO DELETE
	LEA	(NEXT, D0), A1	LOAD ADDRESS OF FORWARD LINK INTO A1
	MOVE.L	(A1), D1	PUT FORWARD LINK VALUE IN D1
	CAS.2	D0:D1, D1:D1, (A0):(A1)	IF STILL POINT TO ENTRY TO BE DELETED, THEN UPDATE HEAD AND FORWARD POINTERS
	BNE	SDLOOP	IF NOT, TRY AGAIN
SDEEMPTY			SUCCESSFUL DELETION, ADDRESS OF DELETED ENTRY IN D0 (MAY BE NULL)

BEFORE DELETING AN ELEMENT:



AFTER DELETING AN ELEMENT:

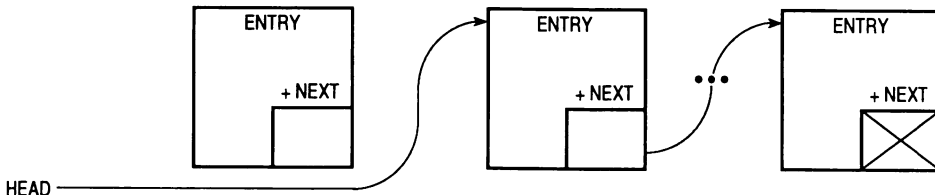
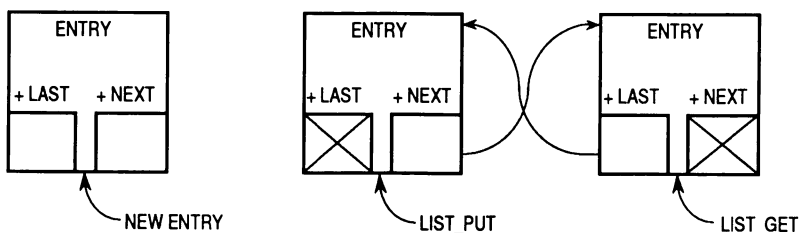


Figure 1-4. Linked List Deletion

LIST-GET, assuming that both of these pointers still contain zero. If not, the BNE instruction branches to the TST instruction at label DILOOP to try again. This time, the BEQ instruction does not branch, and the following MOVE instruction moves the address in D0 to the NEXT pointer of the new element. The CLR instruction clears register D1 to zero, and the MOVE instruction moves the zero into the LAST pointer of the new element. The LEA instruction loads the address of the LAST pointer of the most recently inserted element into register A1. Assuming the LIST-PUT pointer and the pointer in A1 have not been changed, the CAS2 instruction stores the address of the new element into these pointers.

DINSERT	LEA LIST_PUT, A0	(ALLOCATE NEW LIST ENTRY, LOAD ADDRESS INTO A2)
	LEA LIST_GET, A1	LOAD ADDRESS OF HEAD POINTER INTO A0
	MOVE.L A2, D2	LOAD ADDRESS OF TAIL POINTER INTO A1
	MOVE.L (A0), D0	LOAD NEW ENTRY POINTER INTO D2
DILOOP	TST.L D0	LOAD POINTER TO HEAD ENTRY INTO D0
	BEQ DIEMPTY	IS HEAD POINTER NULL, (0 ENTRIES IN LIST)
	MOVE.L D0, (NEXT, A2)	IF SO, WE ONLY TO ESTABLISH POINTERS
	CLR.L D1	PUT HEAD POINTER INTO FORWARD POINTER OF NEW ENTRY
	MOVE.L D1, (LAST, A2)	PUT NULL POINTER VALUE INTO D1
	LEA (LAST, D0), A1	PUT NULL POINTER IN BACKWARD POINTER OF NEW ENTRY
	CAS2.L D0:D1, D2:D2, (A0):(A1)	LOAD BACKWARD POINTER OF OLD HEAD ENTRY INTO A1
	BNE DILOOP	IF WE STILL POINT TO OLD HEAD ENTRY, UPDATE POINTERS
	BRA DIDONE	IF NOT, TRY AGAIN
DIEMPTY	MOVE.L D0, (NEXT, A2)	PUT NULL POINTER IN FORWARD POINTER OF NEW ENTRY
	MOVE.L D0, (LAST, A2)	PUT NULL POINTER IN BACKWARD POINTER OF NEW ENTRY
	CAS2.L D0:D0, D2:D2, (A0):(A1)	IF WE STILL HAVE NO ENTRIES, SET BOTH POINTERS TO THIS ENTRY
	BNE DILOOP	IF NOT, TRY AGAIN
DIDONE		SUCCESSFUL LIST ENTRY INSERTION

BEFORE INSERTING NEW ENTRY:



AFTER INSERTING NEW ENTRY:

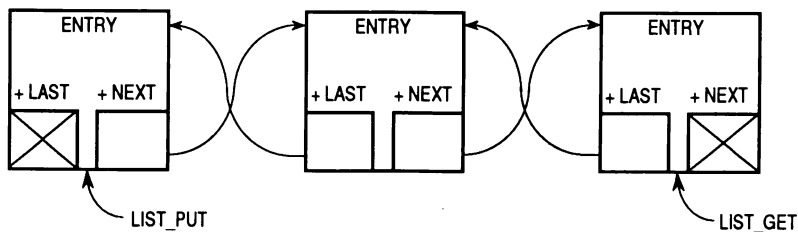
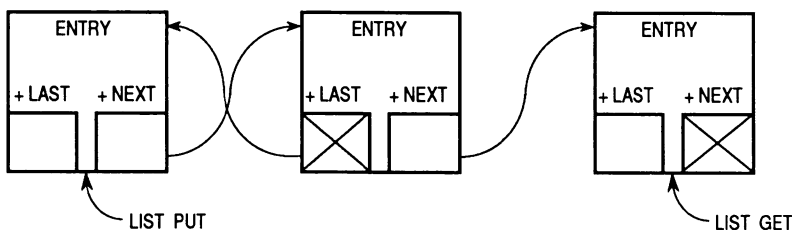


Figure 1-5. Doubly-Linked List Insertion

The code fragment to delete an element from a doubly-linked list is similar (see Figure 1-6). The first two instructions load the effective addresses of pointers LIST_PUT and LIST_GET into registers A0 and A1, respectively. The MOVE instruction at label DDLOOP moves the LIST_GET pointer into register D1. The BEQ instruction that follows branches out of the routine when the

DDELETE	LEA LIST_PUT, A0	GET ADDRESS OF HEAD POINTER IN A0
	LEA LIST_GET, A1	GET ADDRESS OF TAIL POINTER IN A1
DDLOOP	MOVE.L (A1), D1	MOVE TAIL POINTER INTO D1
	BEQ DDONE	IF NO LIST, QUIT
	MOVE.L (LAST, D1), D2	PUT BACKWARD POINTER IN D2
	BEQ DDEEMPTY	IF ONLY ONE ELEMENT, UPDATE POINTERS
	LEA (NEXT, D2), A2	PUT ADDRESS OF FORWARD POINTER IN A2
DDEEMPTY	CRL.L D0	PUT NULL POINTER VALUE IN D0
	CAS2.L D1: D1, D2: D0, (A1): (A2)	IF BOTH POINTERS STILL POINT TO THIS ENTRY, UPDATE THEM
	BNE DDLOOP	IF NOT, TRY AGAIN
	BRA DDONE	
	CAS2.L D1: D1, D2: D2, (A1): (A0)	IF STILL FIRST ENTRY, SET HEAD AND TAIL POINTERS TO NULL
	BNE DDLOOP	IF NOT, TRY AGAIN
DDONE		SUCCESSFUL ENTRY DELETION, ADDRESS OF DELETED ENTRY IN D1 (MAY BE NULL)

BEFORE DELETING NEW ENTRY:



AFTER DELETING NEW ENTRY:

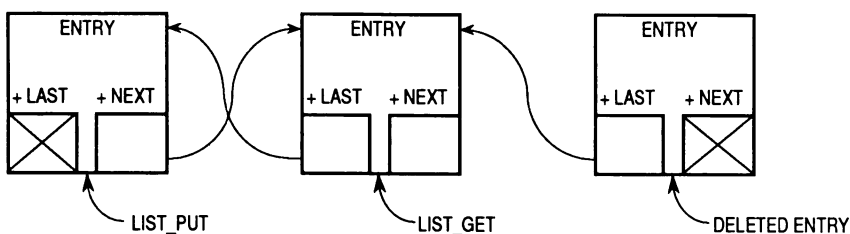


Figure 1-6. Doubly-Linked List Deletion

pointer is zero. The MOVE instruction moves the LAST pointer of the element to be deleted into register D2. Assuming this is not the last element in the list, the Z condition code is not set, and the branch to label DDEEMPTY does not occur. The LEA instruction loads the address of the NEXT pointer of the element at the address in D2 into register A2. The next instruction, a CLR instruction, clears register D0 to zero. The CAS2 instruction compares the address in D1 to the LIST_GET pointer and to the address in register A2. If

the pointers have not been updated, the CAS2 instruction loads the address in D2 into the LIST-GET pointer and zero into the address in register A2.

When the list contains only one element, the routine branches to the CAS2 instruction at label DDEEMPTY after moving a zero pointer value into D2. This instruction checks the addresses in LIST-PUT and LIST-GET to verify that no other routine has inserted another element or deleted the last element. Then the instruction moves zero into both pointers, and the list is empty.

1.2.1.2.2 Nested Subroutine Calls. The LINK instruction pushes an address onto the stack, saves the stack address at which the address is stored, and reserves an area of the stack. Using this instruction in a series of subroutine calls results in a linked list of stack frames.

The UNLK instruction removes a stack frame from the end of the list by loading an address into the stack pointer and pulling the value at that address from the stack. When the operand of the instruction is the address of the link address at the bottom of a stack frame, the effect is to remove the stack frame from the stack and from the linked list.

1.2.1.2.3 Bit Field Instructions. One of the data types provided by the MC68030 is the bit field, consisting of as many as 32 consecutive bits. A bit field is defined by an offset from an effective address and a width value. The offset is a value in the range of -2^{31} through $2^{31} - 1$ from the most significant bit (bit 7) at the effective address. The width is a positive number, 1 through 32. The most significant bit of a bit field is bit 0; the bits number in a direction opposite to the bits of an integer.

The instruction set includes eight instructions that have bit-field operands. The insert bit field (BFINS) instruction inserts a bit field stored in a register into a bit field. The extract bit field signed (BFEXTS) instruction loads a bit field into the least significant bits of a register and extends the sign to the left, filling the register. The extract bit field unsigned (BFEXTU) also loads a bit field, but zero fills the unused portion of the destination register.

The set bit field (BFSET) instruction sets all the bits of a field to ones. The clear bit field (BFCLR) instruction clears a field. The change bit field (BFCHG) instruction complements all the bits in a bit field. These three instructions all test the previous value of the bit field, setting the condition codes accordingly. The test bit field (BFTST) instruction tests the value in the field, setting the condition codes appropriately without altering the bit field. The

find first one in bit field (BFFFO) instruction scans a bit field from bit 0 to the right until it finds a bit set to one and loads the bit offset of the first set bit into the specified data register. If no bits in the field are set, the field offset and the field width are loaded into the register.

An important application of bit-field instructions is the manipulation of the exponent field in a floating-point number. In the IEEE standard format, the most significant bit is the sign bit of the mantissa. The exponent value begins at the next most significant bit position; the exponent field does not begin on a byte boundary. The extract bit field (BFEXTU) instruction and the BFTST instruction are the most useful for this application, but other bit-field instructions can also be used.

Programming of input and output operations to peripherals requires testing, setting, and inserting of bit fields in the control registers of the peripherals. This is another application for bit-field instructions. However, control register locations are not memory locations; therefore, it is not always possible to insert or extract bit fields of a register without affecting other fields within the register.

Another widely used application for bit-field instructions is bit-mapped graphics. Because byte boundaries are ignored in these areas of memory, the field definitions used with bit-field instructions are very helpful.

1.2.1.2.4 Pipeline Synchronization with the NOP Instruction. Although the no operation (NOP) instruction performs no visible operation, it serves an important purpose. It forces synchronization of the integer unit pipeline by waiting for all pending bus cycles to complete. All previous integer instructions and floating-point external operand accesses complete execution before the NOP begins. The NOP instruction does not synchronize the FPU pipeline — floating-point instructions with floating-point register operand destinations can be executing when the NOP begins.

1.2.2 Condition Codes

The CCR portion of the SR contains five bits which are affected by many integer instructions to indicate the results of the instructions. Program and system control instructions use certain combinations of these bits to control program and system flow.

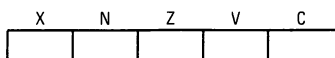
The first four bits represent a condition of the result of a processor operation. The X bit is an operand for multiprecision computations; when it is used, it is set to the value of the carry bit. The carry bit and the multiprecision extend bit are separate in the M68000 Family to simplify programming techniques that use them (refer to Table 1-18 as an example).

The condition codes were developed to meet two criteria:

- Consistency — across instructions, uses, and instances
- Meaningful Results — no change unless it provides useful information

Consistency across instructions means that all instructions that are special cases of more general instructions affect the condition codes in the same way. Consistency across instances means that all instances of an instruction affect the condition codes in the same way. Consistency across uses means that conditional instructions test the condition codes similarly and provide the same results whether the condition codes are set by a compare, test, or move instruction.

In the instruction set definitions, the CCR is shown as follows:



where:

X (extend)

Set to the value of the C bit for arithmetic operations. Otherwise not affected or set to a specified result.

N (negative)

Set if the most significant bit of the result is set. Cleared otherwise.

Z (zero)

Set if the result equals zero. Cleared otherwise.

V (overflow)

Set if arithmetic overflow occurs. This implies that the result cannot be represented in the operand size. Cleared otherwise.

C (carry)

Set if a carry out of the most significant bit of the operand occurs for an addition. Also set if a borrow occurs in a subtraction. Cleared otherwise.

1.2.2.1 CONDITION CODE COMPUTATION. Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Single-operand operations take a destination operand, compute, and store the result in the destination location. Table 1-18 lists each instruction and how it affects the condition code bits.

Table 1-18. Condition Code Computations (Sheet 1 of 2)

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry $Z = Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge \overline{Dm} \vee Sm \wedge \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge \overline{Dm} \vee Sm \wedge \overline{Rm}$ $Z = Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	—	*	*	0	0	
CHK	—	*	U	U	U	
CHK2, CMP2	—	U	?	U	?	$Z = (R = LB) \vee (R = UB)$ $C = (LB < UB) \wedge (R < LB) \vee (R > UB) \vee (UB < LB) \wedge (R > UB) \wedge (R < LB)$
SUB, SUBI, SUBQ	*	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ $Z = Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$
CAS, CAS2, CMP, CMPI, CMPM	—	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
DIVS, DUVI	—	*	*	?	0	V = Division Overflow
MULS, MULU	—	*	*	?	0	V = Multiplication Overflow
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow $Z = Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$
NEG	*	*	*	?	?	$V = Dm \wedge Rm$ $C = Dm \vee Rm$
NEGX	*	*	?	?	?	$V = Dm \wedge Rm$ $C = Dm \vee Rm$ $Z = Z \wedge \overline{Rm} \wedge \dots \wedge \overline{R0}$
BTST, BCHG, BSET, BCLR	—	—	?	—	—	$Z = \overline{Dn}$
BFTST, BFCHG, BFSET, BFCLR	—	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \wedge \overline{DM-1} \wedge \dots \wedge \overline{D0}$
BFEXTS, BFEXTU, BFFFO	—	?	?	0	0	$N = Sm$ $Z = \overline{Sm} \wedge \overline{Sm-1} \wedge \dots \wedge \overline{S0}$
BFINS	—	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \wedge \overline{DM-1} \wedge \dots \wedge \overline{D0}$

Table 1-18. Condition Code Computations (Sheet 2 of 2)

Operations	X	N	Z	V	C	Special Definition
ASL	*	*	*	?	?	$V = D_m \wedge (\overline{D_m - 1} V \dots V \overline{D_m - r}) \vee \overline{D_m} \wedge (D_m - 1 V \dots + D_m - r)$ $C = \overline{D_m - r + 1}$
ASL (R = 0)	—	*	*	0	0	
LSL, ROXL	*	*	*	0	?	$C = D_m - r + 1$
LSR (r = 0)	—	*	*	0	0	
ROXL (r = 0)	—	*	*	0	?	$C = X$
ROL	—	*	*	0	?	$C = D_m - r + 1$
ROL (r = 0)	—	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	$C = D_r - 1$
ASR, LSR (r = 0)	—	*	*	0	0	
ROXR (r = 0)	—	*	*	0	?	$C = X$
ROR	—	*	*	0	?	$C = D_r - 1$
ROR (r = 0)	—	*	*	0	0	

— = Not Affected

U = Undefined, Result Meaningless

? = Other — See Special Definition

* = General Case

X = C

N = Rm

 $Z = \overline{R_m} \wedge \dots \wedge \overline{R_0}$

Sm = Source Operand — Most Significant Bit

Dm = Destination Operand — Most Significant Bit

Rm = Result Operand — Most Significant Bit

R = Register Tested

n = Bit Number

r = Shift Count

LB = Lower Bound

UB = Upper Bound

 \wedge = Boolean AND \vee = Boolean OR

Rm = NOT Rm

1.2.2.2 CONDITIONAL TEST. Table 1-19 lists the condition names, encodings, and tests for the conditional branch and set instructions. The test associated with each condition is a logical formula using the current states of the condition codes. If this formula evaluates to one, the condition is true. If the formula evaluates to zero, the condition is false. For example, the T condition is always true, and the EQ condition is true only if the Z bit condition code is currently true.

1.3 FLOATING-POINT DETAILS

The following paragraphs describe accuracy considerations and conditional tests which can be used to change program flow based on the floating-point conditions. The operation tables in the instruction descriptions are also discussed, followed by details on NaNs and floating-point condition codes.

Table 1-19. Conditional Tests

Mnemonic	Condition	Encoding	Test
T*	True	0000	1
F*	False	0001	0
HI	High	0010	$\overline{C} \cdot \overline{Z}$
LS	Low or Same	0011	$C + Z$
CC(HS)	Carry Clear	0100	\overline{C}
CS(LO)	Carry Set	0101	C
NE	Not Equal	0110	\overline{Z}
EQ	Equal	0111	Z
VC	Overflow Clear	1000	\overline{V}
VS	Overflow Set	1001	V
PL	Plus	1010	\overline{N}
MI	Minus	1011	N
GE	Greater or Equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	Less Than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	Greater Than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	Less or Equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

• = Boolean AND

+ = Boolean OR

\overline{N} = Boolean NOT N

*Not available for the Bcc instruction.

1.3.1 Computational Accuracy

Whenever an attempt is made to represent a real number in a binary format of finite precision, there is a possibility that the number cannot be represented exactly; this is commonly referred to as round-off error. Furthermore, when two inexact numbers are used in a calculation, the error present in each number is reflected and possibly aggravated in the result.

One of the major reasons that the *IEEE Standard for Binary Floating-Point Arithmetic* (ANSI/IEEE Std. 754-1985) was developed was to define the error bounds for calculation of binary floating-point values so that all machines conforming to the standard produce the same results for an operation. The operation must meet the following conditions.

1. same input values,
2. same rounding mode, and
3. same precision.

The IEEE standard specifies not only the format of data items, but also defines:

- the maximum allowable error that may be introduced during a calculation, and
- the manner in which rounding of the result is performed.

The *IEEE Specification for Binary Floating-Point Arithmetic* specifies that the following operations must be supported for each data format: add, subtract, multiply, divide, remainder, square root, integer part, and compare. Conversions between the various data formats are also required. In addition to these arithmetic functions (remainder and integer part are supported in software), the FPU also supports the nontranscendental operations of: absolute value, negate, and test. Since the IEEE specification defines the error bounds to which all calculations are performed, the result obtained by any conforming machine can be predicted exactly for a particular precision and rounding mode. The error bound defined by the IEEE specification is one-half unit in the last place of the destination data format in the round-to-nearest mode, and one unit in the last place in the other rounding modes.

The FPU performs all calculations using a 67-bit mantissa for the intermediate results. The three bits beyond the precision of the extended format allow the FPU to perform all calculations as if to infinite performing calculations in this manner, the final result is always correct for the specified destination data format before rounding is performed (unless an overflow or underflow error occurs). The specified rounding operation then produces a number that is as close as possible to the infinitely-precise-intermediate value and is still representable in the selected precision. An example of how the 67-bit mantissa allows the FPU to meet the error bound of the IEEE specification is as follows:

	Mantissa	1 g r s
Intermediate Result:	x.x.....x00	1 0 0 (Tie Case)
Round-to-Nearest Result:	x.x.....x00	

In this case, the least-significant bit (1) of the rounded result is not incremented, even though the guard bit (g) is set in the intermediate result. The IEEE standard specifies that tie cases should be handled in this manner. Assuming that the destination data format is extended, if the difference between the infinitely precise intermediate result and the round-to-nearest result is calculated, the relative difference is 2^{-64} (the value of the guard bit). This error is equal to one-half of the value of the least significant bit, and is the worst-case error that can be introduced when using the round-to-nearest mode. Thus, the term one-half unit in the last place correctly identifies the

error bound for this operation. This error specification is the relative error present in the result; the absolute error bound is equal to $2^{\text{exponent}} \times 2^{-64}$. An example of the error bound for the other rounding modes is as follows:

	Mantissa	1	g	r	s
Intermediate Result:	x.x.....x00	1	1	1	
Round-to-Zero Result:	x.x.....x00				

In this case, the difference between the infinitely precise result and the rounded result is $2^{-64} + 2^{-65} + 2^{-66}$, which is slightly less than 2^{-63} (the value of the least significant bit). Thus, the error bound for this operation is not more than one unit in the last place. For all of the arithmetic operations, these error bounds are met by the FPU, thus providing accurate and repeatable results.

1.3.2 Conditional Test Definitions

The FPU provides a very simple mechanism for performing conditional tests of the result of any arithmetic floating-point operation. First, the condition code bits in the FPSR are set or cleared at the end of any arithmetic operation or move operation to a single floating-point data register. The condition code bits are always set consistently based on the result of the operation. Second, 32 conditional tests are provided that allow floating-point conditional instructions to test floating-point conditions in exactly the same way as the integer conditional instructions test the integer condition codes. The combination of the consistent setting of the condition code bits and the simple programming of conditional instructions gives the MC68040 a very flexible, high-performance method of altering program flow based on floating-point results.

One important programming consideration is that the inclusion of the NAN data type in the IEEE floating-point number system requires each conditional test to include the NAN condition code bit in its Boolean equation. Because a comparison of a NAN with any other data type is unordered (i.e., it is impossible to determine if a NAN is bigger or smaller than an in-range number), the compare instruction sets the NAN condition code bit when an unordered compare is attempted. All arithmetic instructions also set the NAN bit if the result of an operation is a NAN. The conditional instructions interpret the NAN condition code bit equal to one as the unordered condition.

The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE). Rather, the

opposite condition is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true; whereas, both FBGT and FBLE would be false since unordered fails both of these tests (and sets BSUN). Compiler programmers should be particularly careful of the lack of trichotomy in the floating-point branches since it is common for compilers to invert the sense of conditions.

In the following paragraphs, the conditional tests are described in three main categories:

1. IEEE nonaware tests,
2. IEEE aware tests, and
3. miscellaneous.

The set of IEEE nonaware tests is best used:

1. when porting a program from a system that does not support the IEEE standard to a conforming system, or
2. when generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition).

When using the set of IEEE nonaware tests, the user receives a BSUN exception whenever a branch is attempted and the NAN condition code bit is set, unless the branch is an FBEQ or and FBNE. If the BSUN trap is enabled in the FPCR register, the exception causes a trap. Therefore, the IEEE non-aware program is interrupted if an unexpected condition occurs.

The IEEE aware branch set should be used in programs that contain ordered and unordered conditions by compilers and programmers who are knowledgeable of the IEEE standard. Since the ordered or unordered attribute is explicitly included in the conditional test, the BSUN bit is not set in the status register EXC byte when the unordered condition occurs.

Table 1-20 lists the IEEE nonaware tests. All the conditional tests in Table 1-20, except EQ and NE, set the BSUN bit in the status register exception byte if the NAN condition code bit is set when a conditional instruction is executed.

Table 1-21 lists the IEEE aware tests. None of the conditional tests in Table 1-21 set the BSUN bit in the status register exception byte under any circumstances.

The miscellaneous tests shown in Table 1-22 are not generally used but are implemented for completeness of the set. If the NAN condition code bit is set, T and F do not set the BSUN bit, but SF, ST, SEQ, and SNE do set the BSUN bit.

Table 1-20. IEEE Nonaware Tests

Mnemonic	Definition	Equation	Predicate
EQ	Equal	Z	000001
NE	Not Equal	\bar{Z}	001110
GT	Greater Than	$\overline{N \wedge N \vee Z \vee N}$	010010
NGT	Not Greater Than	$N \wedge N \vee Z \vee N$	011101
GE	Greater Than or Equal	$Z \vee (\overline{N \wedge N \vee N})$	010011
NGE	Not (greater than or equal)	$\overline{N \wedge N \vee (N \wedge \bar{Z})}$	011100
LT	Less Than	$N \wedge (\overline{N \wedge N \vee \bar{Z}})$	010100
NLT	Not Less Than	$\overline{N \wedge N \vee (Z \vee \bar{N})}$	011011
LE	Less Than or Equal	$Z \vee (N \wedge \overline{N \wedge N})$	010101
NLE	Not (less than or equal)	$\overline{N \wedge N \vee (\bar{N} \vee \bar{Z})}$	011010
GL	Greater or Less Than	$\overline{N \wedge N \vee \bar{Z}}$	010110
NGL	Not (greater or less than)	$N \wedge N \vee Z$	011001
GLE	Greater, Less or Equal	$\overline{N \wedge N}$	010111
NGLE	Not (greater, less or equal)	$N \wedge N$	011000

where:

" \vee " = Logical OR

" \wedge " = Logical AND

Table 1-21. IEEE Aware Tests

Mnemonic	Definition	Equation	Predicate
EQ	Equal	Z	000001
NE	Not Equal	\bar{Z}	001110
OGT	Ordered Greater Than	$\overline{N \wedge N \vee Z \vee \bar{N}}$	000010
ULE	Unordered or Less or Equal	$N \wedge N \vee Z \vee N$	001101
OGE	Ordered Greater Than or Equal	$Z \vee (\overline{N \wedge N \vee \bar{N}})$	000011
ULT	Unordered or Less Than	$\overline{N \wedge N \vee (N \wedge \bar{Z})}$	001100
OLT	Ordered Less Than	$N \wedge (\overline{N \wedge N \vee \bar{Z}})$	000100
UGE	Unordered or Greater or Equal	$N \wedge N \vee Z \vee N$	001011
OLE	Ordered Less Than or Equal	$Z \vee (N \wedge \overline{N \wedge N})$	000101
UGT	Unordered or Greater Than	$\overline{N \wedge N \vee (\bar{N} \vee \bar{Z})}$	001010
OGL	Ordered Greater or Less Than	$\overline{N \wedge N \vee \bar{Z}}$	000110
UEQ	Unordered or Equal	$N \wedge N \vee Z$	001001
OR	Ordered	$\overline{N \wedge N}$	000111
UN	Unordered	$N \wedge N$	001000

where:

" \vee " = Logical OR

" \wedge " = Logical AND

Table 1-22. Miscellaneous Tests

Mnemonic	Definition	Equation	Predicate
F	False	False	000000
T	True	True	001111
SF	Signaling False	False	010000
ST	Signaling True	True	011111
SEQ	Signaling Equal	Z	010001
SNE	Signaling Not Equal	\bar{Z}	011110

1.3.3 Operation Tables

An operation table is included for most floating-point instructions. This table lists the result data types for the instruction based on types of input operand(s). For example, Figure 1-7 illustrates the table for the FADD instruction.

Source Destination	In Range		Zero		Infinity	
	+	–	+	–	+	–
In Range	+	–	Add	Add	+ inf	– inf
Zero	+	–	Add	+0.0 0.0 ¹ 0.0 ¹ –0.0	+inf	–inf
Infinity	+	–	+ inf – inf	+ inf – inf	+ inf NAN ²	NAN ² – inf

NOTES:

1. Returns +0.0 in rounding modes RN, RZ, and RP; returns –0.0 in RM.
2. Sets the OPERR bit in the FPSR exception byte.
3. If either operand is a NAN, refer to **1.3.4 NANs** for more information.

Figure 1-7. Operation Table Example (FADD Instruction)

In the example shown in Figure 1-7, the type of the source operand is shown along the top, and the type of the destination operand is shown along the side. In-range numbers are normalized, denormalized, unnormalized real numbers, or integers that are converted to normalized or denormalized extended precision numbers upon entering the FPU.

From Figure 1-7, it can be seen that if both the source and destination operand are positive zero, the result is also a positive zero. For another example, if the source operand is a positive zero and the destination operand is an in-range number, then the ADD algorithm is executed to obtain the result. If a label such as ADD appears in the table, it indicates that the FPU performs the indicated operation and returns the correct result.

A third example of using the tables is when a source operand is plus infinity, and the destination operand is minus infinity. Since the result of such an operation is undefined, a not-a-number (NaN) is returned as the result, and the OPERR bit is set in the floating-point status register (FPSR) exception byte.

1.3.4 NaNs

In addition to the data types covered in the operation tables for each floating-point instruction, NaNs can also be used as inputs to an arithmetic operation. The operation tables do not contain a row and column for NaNs because NaNs are handled the same way in all operations.

If either operand (but not both operands) of an operation is a nonsignaling NaN, then that NaN is returned as the result. If both operands are nonsignaling NaNs, then the destination operand nonsignaling NaN is returned as the result.

If either operand to an operation is a signaling NaN (SNAN), then the SNAN bit is set in the FPSR EXC byte. If the SNAN trap enable bit is set in the floating-point control register (FPCR) ENABLE byte, then the trap is taken and the destination is not modified. If the SNAN trap enable bit is not set, then the SNAN is converted to a nonsignaling NaN (by setting the SNAN bit in the operand to a one), and the operation continues as described in the preceding paragraph for nonsignaling NaNs.

1.3.5 Operation Post Processing

Most operations end with a post processing step. While reading the summary for each instruction, it should be assumed that an instruction performs post processing unless the summary specifically states that the instruction does not do so. The following paragraphs describe post processing in detail.

1.3.5.1 SETTING FLOATING-POINT CONDITION CODES. Unlike the integer arithmetic condition codes, the floating-point condition codes are either not changed by an instruction or are always set in the same way by any instruction. Therefore, it is not necessary to include details of condition code settings for each floating-point instruction in the detailed instruction descriptions. The following paragraphs describe how floating-point condition codes are set for all instructions that modify any condition codes. The four conditions code bits are:

- N—Sign of Mantissa
- Z—Zero
- I—Infinity
- NAN—Not-A-Number

The condition code bits differ slightly from the integer condition codes. The floating-point condition codes are not dependent on the type of operation being performed, but rather, can be set at the end of the operation by examining the result. (The M68000 integer condition codes bits N and Z have this characteristic, but the V and C bits are set differently for different instructions.) At the end of any floating-point operation, the result is inspected, and the condition code bits are set or cleared accordingly. For example, if the result of an operation is a positive normalized number, then all of the condition code bits are set to zero. If the result is a minus infinity, then the N and I bits are set, and the Z and NAN bits are cleared.

1.3.5.2 UNDERFLOW, ROUND, OVERFLOW. During calculation of an arithmetic result, the arithmetic logic unit (ALU) of the FPU has more precision and range than the 80-bit extended precision format. However, the final result of these operations is an extended precision floating-point value. In some cases, an internal result becomes either smaller or larger than can be represented in extended precision. Also, the operation may have generated a larger exponent or more bits of precision than can be represented in the chosen rounding precision. For these reasons, every arithmetic instruction ends by rounding the result and checking for overflow and underflow.

At the completion of an arithmetic operation, the internal result is checked to see if it is too small to be represented as a normalized number in the selected precision. If so, the underflow (UNFL) bit is set in the FPSR EXC byte. It is also denormalized unless denormalization provides a zero value. Denormalizing a number causes a loss of accuracy, but a zero is not returned unless absolutely necessary. If a number is grossly underflowed, the FPU returns a correctly signed zero or the correctly signed smallest denormalized number, depending on the rounding mode in effect.

If no underflow occurs, the internal result is rounded according to the user-selected rounding precision and rounding mode. After rounding, the inexact bit (INEX2) is set appropriately. Lastly, the magnitude of the result is checked to see if it is too large to be represented in the current rounding precision. If so, the overflow (OVFL) bit is set and a correctly signed infinity or correctly signed largest normalized number is returned, depending on the rounding mode in effect.

SECTION 2

INTEGER INSTRUCTIONS

This section contains detailed information about the integer instructions for the M68000 Family. Each instruction is described in detail and the instruction descriptions are arranged in alphabetical order by instruction mnemonic.

Any differences within the M68000 Family of instructions is identified in the instruction. If an instruction only applies to a certain processor or processors, the processor(s) that the instruction pertains to is identified under the title of the instruction. For example:

Test Bit Field and Change (MC68030/MC68040)

If the instruction applies to all the M68000 Family but a processor or processors may use a different instruction field, instruction format, etc., the differences will be identified within the paragraph. For example:

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
-------------	-----	----------------

*Can be used with CPU32 processor

Appendix A provides a listing of all processors and the instructions that applied to them for quick reference.

The MC68HC000 is identical to the MC68000 except for power dissipation, therefore all instructions that apply to the MC68000 also apply to the MC68HC000.

Operation: $\text{Source}_{10} + \text{Destination}_{10} + X \rightarrow \text{Destination}$

Assembler ABCD Dy,Dx

Syntax: ABCD – (Ay), – (Ax)

Attributes: Size = (Byte)

Description: Adds the source operand to the destination operand along with the extend bit, and stores the result in the destination location. The addition is performed using binary coded decimal arithmetic. The operands, which are packed BCD numbers, can be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

X Set the same as the carry bit.

N Undefined.

Z Cleared if the result is nonzero. Unchanged otherwise.

V Undefined.

C Set if a decimal carry was generated. Cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER Rx			1	0	0	0	0	R/M	REGISTER Ry		

Instruction Fields:

Register Rx field — Specifies the destination register:

If R/M = 0, specifies a data register

If R/M = 1, specifies an address register for the predecrement addressing mode

R/M field — Specifies the operand addressing mode:

0 — the operation is data register to data register

1 — the operation is memory to memory

Register Ry field — Specifies the source register:

If R/M = 0, specifies a data register

If R/M = 1, specifies an address register for the predecrement addressing mode

ADD

Add
(M68000 Family)

ADD

2

Operation: Source + Destination \blacklozenge Destination

Assembler ADD <ea>,Dn

Syntax: ADD Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Adds the source operand to the destination operand using binary addition, and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a carry is generated. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE			REGISTER		

ADD

Add (M68000 Family)

ADD

2

Instruction Fields:

Register field — Specifies any of the eight data registers.

Opmode field:

Byte	Word	Long	Operation
000	001	010	$\langle ea \rangle + \langle Dn \rangle \nrightarrow \langle Dn \rangle$
100	101	110	$\langle Dn \rangle + \langle ea \rangle \nrightarrow \langle ea \rangle$

Effective Address Field — Determines addressing mode:

- a. If the location specified is a source operand, all addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Word and Long only.

**Can be used with CPU32.

ADD

Add (M68000 Family)

ADD

- b. If the location specified is a destination operand, only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
-(An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Notes:

1. The Dn mode is used when the destination is a data register; the destination <ea> mode is invalid for a data register.
2. ADDA is used when the destination is an address register. ADDI and ADDQ are used when the source is immediate data. Most assemblers automatically make this distinction.

ADDA

Add Address
(M68000 Family)

ADDA

2

Operation: Source + Destination \rightarrow Destination

Assembler

Syntax: ADDA <ea>, An

Attributes: Size = (Word, Long)

Description: Adds the source operand to the destination address register, and stores the result in the address register. The size of the operation may be specified as word or long. The entire destination address register is used regardless of the operation size.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	REGISTER				OPMODE			EFFECTIVE ADDRESS				
											MODE		REGISTER		

Instruction Fields:

Register field — Specifies any of the eight address registers. This is always the destination.

Opmode field — Specifies the size of the operation:

011 — Word operation. The source operand is sign-extended to a long operand and the operation is performed on the address register using all 32 bits.

111 — Long operation.

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

2

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

ADDI

Add Immediate
(M68000 Family)

ADDI

2

Operation: Immediate Data + Destination \rightarrow Destination

Assembler

Syntax: ADDI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Adds the immediate data to the destination operand, and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a carry is generated. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
0	0	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS											
										MODE			REGISTER								
WORD DATA (16 BITS)								BYTE DATA (8 BITS)													
LONG DATA (32 BITS)																					

Instruction Fields:

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination operand.

Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(d ₈ ,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Immediate field — (data immediately following the instruction):

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

ADDQ

Add Quick
(M68000 Family)

ADDQ

2

Operation: Immediate Data + Destination \rightarrow Destination

Assembler

Syntax: ADDQ #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Adds an immediate value of one to eight to the operand at the destination location. The size of the operation may be specified as byte, word, or long. Word and long operations are also allowed on the address registers. When adding to address registers, the condition codes are not altered, and the entire destination address register is used regardless of the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow occurs. Cleared otherwise.
- C Set if a carry occurs. Cleared otherwise.

The condition codes are not affected when the destination is an address register.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	DATA			0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Data field — Three bits of immediate data, 0–7 (with the immediate value zero representing a value of eight).

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination location.

Only alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Word and Long only.

**Can be used with CPU32.

ADDX

Add Extended
(M68000 Family)

ADDX

2

Operation: Source + Destination + X \rightarrow Destination

Assembler ADDX Dy,Dx

Syntax: ADDX – (Ay), – (Ax)

Attributes: Size = (Byte, Word, Long)

Description: Adds the source operand to the destination operand along with the extend bit and stores the result in the destination location. The operands can be addressed in two different ways:

1. Data register to data register: The data registers specified in the instruction contain the operands.
2. Memory to memory: The address registers specified in the instruction address the operands using the predecrement addressing mode.

The size of the operation can be specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit.
N Set if the result is negative. Cleared otherwise.
Z Cleared if the result is nonzero. Unchanged otherwise.
V Set if an overflow occurs. Cleared otherwise.
C Set if a carry is generated. Cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

ADDX

Add Extended
(M68000 Family)

ADDX

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	REGISTER Rx			1	SIZE		0	0	R/M	REGISTER Ry		

2

Instruction Fields:

Register Rx field — Specifies the destination register:

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

R/M field — Specifies the operand address mode:

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Ry field — Specifies the source register:

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

AND

AND Logical (M68000 Family)

AND

2

Operation: Source \wedge Destination \rightarrow Destination

Assembler AND <ea>,Dn

Syntax: AND Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Performs an AND operation of the source operand with the destination operand and stores the result in the destination location. The size of the operation can be specified as byte, word, or long. The contents of an address register may not be used as an operand.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the result is set. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

Register field — Specifies any of the eight data registers.

Opmode field:

Byte	Word	Long	Operation
000	001	010	((ea)) \wedge ((Dn)) \rightarrow Dn
100	101	110	((Dn)) \wedge ((ea)) \rightarrow ea

Effective Address field — Determines addressing mode:

- a. If the location specified is a source operand only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

- b. If the location specified is a destination operand only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Notes:

1. The Dn mode is used when the destination is a data register; the destination <ea> mode is invalid for a data register.
2. Most assemblers use ANDI when the source is immediate data.

ANDI

AND Immediate (M68000 Family)

ANDI

Operation: Immediate Data \wedge Destination \rightarrow Destination

Assembler

Syntax: ANDI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Performs an AND operation of the immediate data with the destination operand and stores the result in the destination location. The size of the operation can be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the result is set. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
0	0	0	0	0	0	1	0	SIZE		EFFECTIVE ADDRESS											
										MODE		REGISTER									
WORD DATA (16 BITS)								BYTE DATA (8 BITS)													
LONG DATA (32 BITS)																					

Instruction Fields:

Size field — Specifies the size of the operation:

- 00 — Byte operation
- 01 — Word operation.
- 10 — Long operation.

Effective Address field — Specifies the destination operand.

Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

{bd,An,Xn}*	110	reg. number:An
{[bd,An,Xn],od}	110	reg. number:An
{[bd,An],Xn,od}	110	reg. number:An

{bd,PC,Xn}*	—	—
{[bd,PC,Xn],od}	—	—
{[bd,PC],Xn,od}	—	—

*Can be used with CPU32.

Immediate field — (data immediately following the instruction):

- If size = 00, the data is the low-order byte of the immediate word.
- If size = 01, the data is the entire immediate word.
- If size = 10, the data is the next two immediate words.

ANDI to CCR

AND Immediate to Condition Codes
(M68000 Family)

ANDI to CR

2

Operation: Source \wedge CCR \rightarrow CCR

Assembler

Syntax: ANDI #(data),CCR

Attributes: Size = (Byte)

Description: Performs an AND operation of the immediate operand with the condition codes and stores the result in the low-order byte of the status register.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X Cleared if bit 4 of immediate operand is zero. Unchanged otherwise.

N Cleared if bit 3 of immediate operand is zero. Unchanged otherwise.

Z Cleared if bit 2 of immediate operand is zero. Unchanged otherwise.

V Cleared if bit 1 of immediate operand is zero. Unchanged otherwise.

C Cleared if bit 0 of immediate operand is zero. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	BYTE DATA (8 BITS)							

ASL,ASR

Arithmetic Shift
(M68000 Family)

ASL,ASR

2

Operation: Destination Shifted by <count> ➔ Destination

Assembler ASd Dx,Dy

Syntax: ASd #<data>,Dy

ASd <ea>

where d is direction, L or R

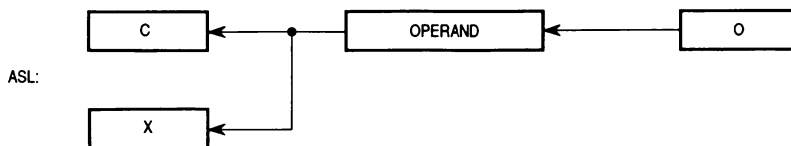
Attributes: Size = (Byte, Word, Long)

Description: Arithmetically shifts the bits of the operand in the direction (L or R) specified. The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register may be specified in two different ways:

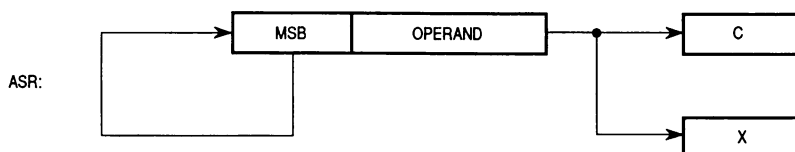
1. Immediate — The shift count is specified in the instruction (shift range, 1–8).
2. Register — The shift count is the value in the data register specified in instruction modulo 64.

The size of the operation can be specified as byte, word, or long. An operand in memory can be shifted one bit only, and the operand size is restricted to a word.

For ASL, the operand is shifted left; the number of positions shifted is the shift count. Bits shifted out of the high-order bit go to both the carry and the extend bits; zeros are shifted into the low-order bit. The overflow bit indicates if any sign changes occur during the shift.



For ASR, the operand is shifted right; the number of positions shifted is the shift count. Bits shifted out of the low-order bit go to both the carry and the extend bits; the sign-bit (MSB) is shifted into the high-order bit.



Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if the most significant bit is changed at any time during the shift operation. Cleared otherwise.
- C Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.

Instruction Format (Register Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/REGISTER		dr	SIZE		i/r	0	0	REGISTER			

Instruction Fields (Register Shifts):

Count/Register field — Specifies shift count or register that contains the shift count:

If $i/r = 0$, this field contains the shift count. The values 1–7 represent counts of 1–7; value of zero represents a count of eight.

If $i/r = 1$, this field specifies the data register that contains the shift count (modulo 64).

ASL,ASR

Arithmetic Shift (M68000 Family)

ASL,ASR

dr field — Specifies the direction of the shift:

0 — Shift right.

1 — Shift left.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

i/r field:

If i/r = 0, specifies immediate shift count.

If i/r = 1, specifies register shift count.

Register field — Specifies a data register to be shifted.

Instruction Format (Memory Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	EFFECTIVE ADDRESS MODE REGISTER					

Instruction Fields (Memory Shifts):

dr field — Specifies the direction of the shift:

0 — Shift right.

1 — Shift left.

Effective Address field — Specifies the operand to be shifted.

Only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Bcc

Branch Conditionally (M68000 Family)

Bcc

Operation: If (condition true) then $PC + d \rightarrow PC$

Assembler

Syntax: Bcc <label>

Attributes: Size = (Byte, Word, Long*)
*(MC68020/MC68030/MC68040 only)

Description: If the specified condition is true, program execution continues at location $(PC) + \text{displacement}$. The PC contains the address of the instruction word of the Bcc instruction plus two. The displacement is a twos complement integer that represents the relative distance in bytes from the current PC to the destination PC. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (\$FF), the 32-bit displacement (long word immediately following the instruction) is used. Condition code cc specifies one of the following conditions:

CC	carry clear	0100	\overline{C}
CS	carry set	0101	C
EQ	equal	0111	Z
GE	greater or equal	1100	$N \cdot V + \overline{N \cdot V}$
GT	greater than	1110	$\overline{N \cdot V \cdot Z} + \overline{N \cdot V \cdot \overline{Z}}$
HI	high	0010	$C \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N \cdot V}$
LS	low or same	0011	$C + \overline{Z}$
LT	less than	1101	$\overline{N \cdot V} + N \cdot V$
MI	minus	1011	\overline{N}
NE	not equal	0110	\overline{Z}
PL	plus	1010	\overline{N}
VC	overflow clear	1000	\overline{V}
VS	overflow set	1001	V

Condition Codes:

Not affected.

Bcc**Branch Conditionally**
(M68000 Family)**Bcc****Instruction Format:**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	CONDITION				8-BIT DISPLACEMENT							
16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00															
32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF															

2**Instruction Fields:**

Condition field — The binary code for one of the conditions listed in the table.

8-Bit Displacement field — Twos complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed if the condition is met.

16-Bit Displacement field — Used for the displacement when the 8-bit displacement field contains \$00.

32-Bit Displacement field — Used for the displacement when the 8-bit displacement field contains \$FF.

NOTE

A branch to the immediately following instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains \$00 (zero offset).

Operation: $\sim(\langle\text{number}\rangle \text{ of Destination}) \nrightarrow Z;$
 $\sim(\langle\text{number}\rangle \text{ of Destination}) \nrightarrow \langle\text{bit number}\rangle \text{ of Destination}$

Assembler BCHG Dn,<ea>
Syntax: BCHG #<data>,<ea>

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately, then inverts the specified bit in the destination. When the destination is a data register, any of the 32 bits can be specified by the modulo 32-bit number. When the destination is a memory location, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation may be specified in either of two ways:

1. Immediate — The bit number is specified in a second word of the instruction.
2. Register — The specified data register contains the bit number.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

- X Not affected.
 N Not affected.
 Z Set if the bit tested is zero. Cleared otherwise.
 V Not affected.
 C Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	0	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register that contains the bit number.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

**Can be used with CPU32.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	0	BIT NUMBER							

2

Instruction Fields (Bit Number Static):

Effective Address field — Specifies the destination location.

Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(d ₈ ,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

**Can be used with CPU32.

Bit Number field — Specifies the bit number.

BCLR

Test a Bit and Clear (M68000 Family)

BCLR

2

Operation: $\sim(\langle \text{bit number} \rangle \text{ of Destination}) \nabla Z;$
 $0 \nabla \langle \text{bit number} \rangle \text{ of Destination}$

Assembler BCLR Dn,⟨ea⟩

Syntax: BCLR #⟨data⟩,⟨ea⟩

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately, then clears the specified bit in the destination. When a data register is the destination, any of the 32 bits can be specified by a modulo 32-bit number. When a memory location is the destination, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation can be specified in either of two ways:

1. Immediate — The bit number is specified in a second word of the instruction.
2. Register — The specified data register contains the bit number.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

- X Not affected.
- N Not affected.
- Z Set if the bit tested is zero. Cleared otherwise.
- V Not affected.
- C Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	1	0	EFFECTIVE ADDRESS MODE REGISTER					

2

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register that contains the bit number.

Effective Address field — Specifies the destination location.

Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

**Can be used with CPU32.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	0	BIT NUMBER							

Instruction Fields (Bit Number Static):

Effective Address field — Specifies the destination location.

Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

**Can be used with CPU32.

Bit Number field — Specifies the bit number.

BFCHG

Test Bit Field and Change
(MC68020/MC68030/MC68040)

BFCHG

Operation: $\sim(\langle \text{bit field} \rangle \text{ of Destination}) \uparrow \langle \text{bit field} \rangle \text{ of Destination}$

Assembler

Syntax: BFCHG $\langle \text{ea} \rangle \{ \text{offset:width} \}$

Attributes: Unsized

Description: Sets the condition codes according to the value in a bit field at the specified effective address, then complements the field.

A field offset and a field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [3:4] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; an operand value in the range 1–31 specifies a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFCLR

Test Bit Field and Clear
(MC68020/MC68030/MC68040)

BFCLR

Operation: 0 ∇ <bit field> of Destination

Assembler

Syntax: BFCLR <ea>{offset:width}

Attributes: Unsized

Description: Sets condition codes according to the value in a bit field at the specified effective address, and clears the field.

The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or alterable control addressing modes are allowed, as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [3:4] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFEXTS

Extract Bit Field Signed
(MC68020/MC68030/MC68040)

BFEXTS

Operation: <bit field> of Source ♦ Dn

Assembler

Syntax: BFEXTS <ea>{offset:width},Dn

Attributes: Unsized

Description: Extracts a bit field from the specified effective address location, sign extends to 32 bits, and loads the result into the destination data register.

The field offset and field width select the bit field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	1	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	REGISTER			Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	—	—
(An) +	—	—			
– (An)	—	—			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

Register field — Specifies the destination register.

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [4:3] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFEXTU

Extract Bit Field Unsigned
(MC68020/MC68030/MC68040)

BFEXTU

Operation: ⟨bit offset⟩ of Source ↗ Dn

Assembler

Syntax: BFEXTU ⟨ea⟩{offset:width},Dn

Attributes: Unsize

Description: Extracts a bit field from the specified effective address location, zero extends to 32 bits, and loads the results into the destination data register.

The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the most significant bit of the source field is set. Cleared otherwise.
- Z Set if all bits of the field are zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	1	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	REGISTER			Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(d ₈ ,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register field — Specifies the destination data register.

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [4:3] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFFFO

Find First One in Bit Field
(MC68020/MC68030/MC68040)

BFFFO

Operation: ⟨bit offset⟩ of Source Bit Scan → Dn

Assembler

Syntax: BFFFO ⟨ea⟩{offset:width},Dn

Attributes: Unsize

Description: Searches the source operand for the most significant bit that is set to a value of one. The bit offset of that bit (the bit offset in the instruction plus the offset of the first one bit) is placed in Dn. If no bit in the bit field is set to one, the value in Dn is the field offset plus the field width. The instruction sets the condition codes according to the bit field value.

The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	REGISTER			Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register field — Specifies the destination data register operand.

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [4:3] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFINS

Insert Bit Field
(MC68020/MC68030/MC68040)

BFINS

Operation: Dn \leftarrow <bit field> of Destination

Assembler

Syntax: BFINS Dn,<ea>{offset:width}

Attributes: Unsized

Description: Inserts a bit field taken from the low-order bits of the specified data register into a bit field at the effective address location. The instruction sets the condition codes according to the inserted value.

The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	REGISTER			Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Register field — Specifies the source data register operand.

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [4:3] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFSET

Test Bit Field and Set
(MC68020/MC68030/MC68040)

BFSET

Operation: 1s \blacktriangledown <bit field> of Destination

Assembler

Syntax: BFSET <ea>{offset:width}

Attributes: Unsized

Description: Sets the condition codes according to the value in a bit field at the specified effective address, then sets each bit in the field.

The field offset and the field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [4:3] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand; operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

BFTST

Test Bit Field
(MC68020/MC68030/MC68040)

BFTST

Operation: ⟨bit field⟩ of Destination

Assembler

Syntax: BFTST ⟨ea⟩{offset:width}

Attributes: Unsize

Description: Sets the condition codes according to the value in a bit field at the specified effective address location.

The field offset and field width select the field. The field offset specifies the starting bit of the field. The field width determines the number of bits in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the field is set. Cleared otherwise.

Z Set if all bits of the field are zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	0	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	Do	OFFSET					Dw	WIDTH				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	—	—
(An) +	—	—			
-(An)	—	—			
(d ₁₆ ,An)	101	reg. number:An	(d ₁₆ ,PC)	111	010
(d ₈ ,An,Xn)	110	reg. number:An	(d ₈ ,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

Do field — Determines how the field offset is specified.

0 — The Offset field contains the bit field offset.

1 — Bits [8:6] of the extension word specify a data register that contains the offset; bits [10:9] are zero.

Offset field — Specifies the field offset, depending on Do.

If Do = 0, the Offset field is an immediate operand; the operand value is in the range of 0–31.

If Do = 1, the Offset field specifies a data register that contains the offset. The value is in the range of -2^{31} to $2^{31} - 1$.

Dw field — Determines how the field width is specified.

0 — The Width field contains the bit field width.

1 — Bits [2:0] of the extension word specify a data register that contains the width; bits [4:3] are zero.

Width field — Specifies the field width, depending on Dw.

If Dw = 0, the Width field is an immediate operand, operand values in the range of 1–31 specify a field width of 1–31, and a value of zero specifies a width of 32.

If Dw = 1, the Width field specifies a data register that contains the width. The value is modulo 32; values of 1–31 specify field widths of 1–31, and a value of zero specifies a width of 32.

Operation: Run breakpoint acknowledge cycle;
TRAP as illegal instruction

Assembler

Syntax: BKPT #(data)

Attributes: Unsized

Description: For the **MC68010**, a breakpoint acknowledge bus cycle is run with function codes driven high and zeros on all address lines. Whether the breakpoint acknowledge bus cycle is terminated with \overline{DTACK} , \overline{BERR} , or \overline{VPA} , the processor always takes an illegal instruction exception. During exception processing, a debug monitor can distinguish different software breakpoints by decoding the field in the BKPT instruction. For the **MC68000** and **MC68008** the breakpoint cycle is not run but an illegal instruction exception is taken.

For the **MC68020**, **MC68030**, and **CPU32**, a breakpoint acknowledge bus cycle is executed with the immediate data (value 0–7) on bits 2–4 of the address bus and zeros on bits 0 and 1 of the address bus. The breakpoint acknowledge bus cycle accesses the CPU space, addressing type 0, and provides the breakpoint number specified by the instruction on address lines A2–A4. If the external hardware terminates the cycle with \overline{DSACKx} or \overline{STERM} , the data on the bus (an instruction word) is inserted into the instruction pipe and is executed after the breakpoint instruction. The breakpoint instruction requires a word to be transferred so, if the first bus cycle accesses an 8-bit port, a second bus cycle is required. If the external logic terminates the breakpoint acknowledge bus cycle with \overline{BERR} (i.e., no instruction word available) the processor takes an illegal instruction exception.

For the **MC68040**, this instruction executes a breakpoint acknowledge bus cycle. Regardless of the cycle termination, the MC68040 takes an illegal instruction exception.

For more information on the breakpoint instruction refer to the appropriate processor user's manual on **BUS OPERATION**.

This instruction supports breakpoints for debug monitors and real-time hardware emulators.

BKPT

Breakpoint (MC68010/MC68020/MC68030/MC68040/CPU32)

BKPT

2

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	VECTOR		

Instruction Fields:
Vector field — Contains the immediate data, a value in the range of 0–7. This is the breakpoint number.

BRA

Branch Always
(M68000 Family)

BRA

Operation: PC + d ↗ PC

Assembler

Syntax: BRA <label>

Attributes: Size = (Byte, Word, Long*)
*(MC68020/MC68030/MC68040 only)

Description: Program execution continues at location (PC) + displacement. The PC contains the address of the instruction word of the BRA instruction plus two. The displacement is a twos complement integer that represents the relative distance in bytes from the current PC to the destination PC. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (\$FF), the 32-bit displacement (long word immediately following the instruction) is used.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8-BIT DISPLACEMENT							
16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00															
32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF															

Instruction Fields:

8-Bit Displacement field — Twos complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.

16-Bit Displacement field — Used for a larger displacement when the 8-bit displacement is equal to \$00.

32-Bit Displacement field — Used for a larger displacement when the 8-bit displacement is equal to \$FF.

NOTE

A branch to the immediately following instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains \$00 (zero offset).

Operation: $\sim(\langle \text{bit number} \rangle \text{ of Destination}) \nabla Z;$
 $1 \nabla \langle \text{bit number} \rangle \text{ of Destination}$

Assembler BSET Dn,<ea>

Syntax: BSET #<data>,<ea>

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately. Then sets the specified bit in the destination operand. When a data register is the destination, any of the 32 bits can be specified by a modulo 32-bit number. When a memory location is the destination, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation can be specified in either of two ways:

1. Immediate — The bit number is specified in the second word of the instruction.
2. Register — The specified data register contains the bit number.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

X Not affected.

N Not affected.

Z Set if the bit tested is zero. Cleared otherwise.

V Not affected.

C Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register that contains the bit number.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
— (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

**Can be used with CPU32.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	BIT NUMBER								

Instruction Fields (Bit Number Static):

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

**Can be used with CPU32.

Bit Number field — Specifies the bit number.

BSR

Branch to Subroutine (M68000 Family)

BSR

Operation: $SP - 4 \nrightarrow SP$; $PC \nrightarrow (SP)$; $PC + d \nrightarrow PC$

Assembler

Syntax: BSR <label>

Attributes: Size = (Byte, Word, Long*)
*(MC68020/MC68030/MC68040 only)

Description: Pushes the long word address of the instruction immediately following the BSR instruction onto the system stack. The PC contains the address of the instruction word plus two. Program execution then continues at location $(PC) + \text{displacement}$. The displacement is a twos complement integer that represents the relative distance in bytes from the current PC to the destination PC. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (\$FF), the 32-bit displacement (long word immediately following the instruction) is used.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8-BIT DISPLACEMENT							
16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00															
32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF															

Instruction Fields:

- 8-Bit Displacement field — Twos complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.
- 16-Bit Displacement field — Used for a larger displacement when the 8-bit displacement is equal to \$00.
- 32-Bit Displacement field — Used for a larger displacement when the 8-bit displacement is equal to \$FF.

NOTE

A branch to the immediately following instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains \$00 (zero offset).

Operation: – ((bit number) of Destination) ∇ Z;

Assembler BTST Dn,<ea>

Syntax: BTST #<data>,<ea>

Attributes: Size = (Byte, Long)

Description: Tests a bit in the destination operand and sets the Z condition code appropriately. When a data register is the destination, any of the 32 bits can be specified by a modulo 32-bit number. When a memory location is the destination, the operation is a byte operation, and the bit number is modulo 8. In all cases, bit zero refers to the least significant bit. The bit number for this operation can be specified in either of two ways:

1. Immediate — The bit number is specified in a second word of the instruction.
2. Register — The specified data register contains the bit number.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

X Not affected.

N Not affected.

Z Set if the bit tested is zero. Cleared otherwise.

V Not affected.

C Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register that contains the bit number.

Effective Address field — Specifies the destination location. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Long only; all others are byte only.

**Can be used with CPU32.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	0	BIT NUMBER							

Instruction Fields (Bit Number Static):

Effective Address field — Specifies the destination location. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Bit Number field — Specifies the bit number.

Operation: Save current module state on stack;
Load new module state from destination

Assembler

Syntax: CALLM #<data>, <ea>

Attributes: Unsized

Description: The effective address of the instruction is the location of an external module descriptor. A module frame is created on the top of the stack, and the current module state is saved in the frame. The immediate operand specifies the number of bytes of arguments to be passed to the called module. A new module state is loaded from the descriptor addressed by the effective address.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	0	ARGUMENT COUNT							

Instruction Fields:

Effective Address field — Specifies the address of the module descriptor. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Argument Count field — Specifies the number of bytes of arguments to be passed to the called module. The 8-bit field can specify from 0 to 255 bytes of arguments. The same number of bytes is removed from the stack by the RTM instruction.

Operation: CAS Destination — Compare Operand ♦ cc;
if Z, Update Operand ♦ Destination
else Destination ♦ Compare Operand
CAS2 Destination 1 — Compare 1 ♦ cc;
if Z, Destination 2 — Compare 2 ♦ cc
if Z, Update 1 ♦ Destination 1; Update 2 ♦ Destination 2
else Destination 1 ♦ Compare 1; Destination 2 ♦ Compare 2

Assembler CAS Dc,Du,<ea>

Syntax: CAS2 Dc1:Dc2,Du1:Du2,(Rn1):(Rn2)

Attributes: Size = (Byte*, Word, Long)

Description: CAS compares the effective address operand to the compare operand (Dc). If the operands are equal, the instruction writes the update operand (Du) to the effective address operand; otherwise, the instruction writes the effective address operand to the compare operand (Dc).

CAS2 compares memory operand 1 (Rn1) to compare operand 1 (Dc1). If the operands are equal, the instruction compares memory operand 2 (Rn2) to compare operand 2 (Dc2). If these operands are also equal, the instruction writes the update operands (Du1 and Du2) to the memory operands (Rn1 and Rn2). If either comparison fails, the instruction writes the memory operands (Rn1 and Rn2) to the compare operands (Dc1 and Dc2).

Both operations access memory using locked or read-modify-write transfer sequences. This provides a means of synchronizing several processors.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected.

N Set if the result is negative. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Set if an overflow is generated. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

*CAS2 cannot use byte operands

Instruction Format: (CAS):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	SIZE		0	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	0	0	0	Du			0	0	0	Dc		

Instruction Fields:

Size field — Specifies the size of the operation.

01 — Byte operation.

10 — Word operation.

11 — Long operation.

Effective Address field — Specifies the location of the memory operand. Only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Du field — Specifies the data register that contains the update value to be written to the memory operand location if the comparison is successful.

Dc field — Specifies the data register that contains the value to be compared to the memory operand.

Instruction Format (CAS2):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	SIZE			0	1	1	1	1	1	0	0
D/A1	Rn1			0	0	0	Du1			0	0	0	Dc1		
D/A2	Rn2			0	0	0	Du2			0	0	0	Dc2		

Instruction Fields:

Size field — Specifies the size of the operation.

10 — Word operation.

11 — Long operation.

D/A1, D/A2 fields — Specify whether Rn1 and Rn2 reference data or address registers, respectively.

0 — The corresponding register is a data register.

1 — The corresponding register is an address register.

Rn1, Rn2 fields — Specify the numbers of the registers that contain the addresses of the first and second memory operands, respectively. If the operands overlap in memory, the results of any memory update are undefined.

Du1, Du2 fields — Specify the data registers that contain the update values to be written to the first and second memory operand locations if the comparison is successful.

Dc1, Dc2 fields — Specify the data registers that contain the test values to be compared to the first and second memory operands, respectively. If Dc1 and Dc2 specify the same data register and the comparison fails, memory operand 1 is stored in the data register.

NOTES

The CAS and CAS2 instructions can be used to perform secure update operations on system control data structures in a multiprocessing environment.

In the MC68040 if the operands are not equal, the destination or destination 1 operand is written back to memory to complete the locked access for CAS or CAS2, respectively.

Operation: If $D_n < 0$ or $D_n > \text{Source}$ then TRAP

Assembler

Syntax: CHK <ea>,Dn

Attributes: Size = (Word, Long*)
*(MC68020/MC68030/MC68040 only)

Description: Compares the value in the data register specified in the instruction to zero and to the upper bound (effective address operand). The upper bound is a twos complement integer. If the register value is less than zero or greater than the upper bound, a CHK instruction exception, vector number 6, occurs.

Condition Codes:

X	N	Z	V	C
—	*	U	U	U

X Not affected.

N Set if $D_n < 0$; cleared if $D_n > \text{effective address operand}$.
Undefined otherwise.

Z Undefined.

V Undefined.

C Undefined.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	REGISTER			SIZE		0	EFFECTIVE ADDRESS					
											MODE		REGISTER		

Instruction Fields:

Register field — Specifies the data register that contains the value to be checked.

Size field — Specifies the size of the operation.

11 — Word operation.

10 — Long operation.

Effective Address field — Specifies the upper bound operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(d8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(d8,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Operation: If $R_n < \text{lower bound}$ or
 $R_n > \text{upper bound}$
 then TRAP

Assembler

Syntax: CHK2 (ea),Rn

Attributes: Size = (Byte, Word, Long)

Description: Compares the value in Rn to each bound. The effective address contains the bounds pair: the lower bound followed by the upper bound. For signed comparisons, the arithmetically smaller value should be used as the lower bound. For unsigned comparisons, the logically smaller value should be the lower bound.

The size of the data and the bounds can be specified as byte, word, or long. If Rn is a data register and the operation size is byte or word, only the appropriate low-order part of Rn is checked. If Rn is an address register and the operation size is byte or word, the bounds operands are sign extended to 32 bits and the resultant operands are compared to the full 32 bits of An.

If the upper bound equals the lower bound, the valid range is a single value. If the register value is less than the lower bound or greater than the upper bound, a CHK instruction exception, vector number 6, occurs.

Condition Codes:

X	N	Z	V	C
—	U	*	U	*

X Not affected.

N Undefined.

Z Set if Rn is equal to either bound. Cleared otherwise.

V Undefined.

C Set if Rn is out of bounds. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS					
D/A				REGISTER		1	0	0	0	0	0	0	0	0	0
										MODE		REGISTER			

Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the location of the bounds operands. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

D/A field — Specifies whether an address register or data register is to be checked.

0 — Data register.

1 — Address register.

Register field — Specifies the address or data register that contains the value to be checked.

CLR

Clear an Operand (M68000 Family)

CLR

Operation: 0 ♦ Destination

Assembler

Syntax: CLR (ea)

Attributes: Size = (Byte, Word, Long)

Description: Clears the destination operand to zero. The size of the operation may be specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	0	1	0	0

- X Not affected.
- N Always cleared.
- Z Always set.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(d ₈ ,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

NOTE

In the MC68000 and MC68008 a memory location is read before it is cleared.

CMP

Compare
(M68000 Family)

CMP

Operation: Destination — Source ∇ cc

Assembler

Syntax: CMP <ea>, Dn

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination data register and sets the condition codes according to the result; the data register is not changed. The size of the operation can be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

- X Not affected.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow occurs. Cleared otherwise.
- C Set if a borrow occurs. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Register field — Specifies the destination data register.

Opmode field:

Byte	Word	Long	Operation
000	001	010	((Dn)) – ((ea))

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Word and Long only.
**Can be used with CPU32.

NOTE

CMPA is used when the destination is an address register. CMPI is used when the source is immediate data. CMPM is used for memory-to-memory compares. Most assemblers automatically make the distinction.

Operation: Destination – Source

Assembler

Syntax: CMPA <ea>, An

Attributes: Size = (Word, Long)

Description: Subtracts the source operand from the destination address register and sets the condition codes according to the result; the address register is not changed. The size of the operation can be specified as word or long. Word length source operands are sign extended to 32 bits for comparison.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected.

N Set if the result is negative. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Set if an overflow is generated. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Register field — Specifies the destination address register.
Opmode field — Specifies the size of the operation:

011 — Word operation. The source operand is sign extended to a long operand and the operation is performed on the address register using all 32 bits.
111 — Long operation.

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	001	reg. number:An	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

CMPI

Compare Immediate (M68000 Family)

CMPI

Operation: Destination – Immediate Data

Assembler

Syntax: CMPI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the immediate data from the destination operand and sets the condition codes according to the result; the destination location is not changed. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected.

N Set if the result is negative. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Set if an overflow occurs. Cleared otherwise.

C Set if a borrow occurs. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

Instruction Fields:

Size field — Specifies the size of the operation:

- 00 — Byte operation.
- 01 — Word operation.
- 10 — Long operation.

Effective Address field — Specifies the destination operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(d8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	111	010
(d8,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An,Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Immediate field — (Data immediately following the instruction):

- If size = 00, the data is the low-order byte of the immediate word.
- If size = 01, the data is the entire immediate word.
- If size = 10, the data is the next two immediate words.

Operation: Destination — Source ∇ cc

Assembler

Syntax: CMPM (Ay) + ,(Ax) +

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination operand and sets the condition codes according to the results; the destination location is not changed. The operands are always addressed with the postincrement addressing mode, using the address registers specified in the instruction. The size of the operation may be specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected.

N Set if the result is negative. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Set if an overflow is generated. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER Ax			1	SIZE		0	0	1	REGISTER Ay		

Instruction Fields:

Register Ax field — (always the destination) Specifies an address register in the postincrement addressing mode.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Register Ay field — (always the source) Specifies an address register in the postincrement addressing mode.

CMP2

Compare Register Against Bounds (MC68020/MC68030/MC68040/CPU32)

CMP2

2

Operation: Compare $R_n < \text{lower-bound}$ or
 $R_n > \text{upper-bound}$
and Set Condition Codes

Assembler

Syntax: CMP2 <ea>,Rn

Attributes: Size = (Byte, Word, Long)

Description: Compares the value in R_n to each bound. The effective address contains the bounds pair: the lower bound followed by the upper bound. For signed comparisons, the arithmetically smaller value should be used as the lower bound. For unsigned comparisons, the logically smaller value should be the lower bound.

The size of the data and the bounds can be specified as byte, word, or long. If R_n is a data register and the operation size is byte or word, only the appropriate low-order part of R_n is checked. If R_n is an address register and the operation size is byte or word, the bounds operands are sign extended to 32 bits and the resultant operands are compared to the full 32 bits of A_n .

If the upper bound equals the lower bound, the valid range is a single value.

NOTE

This instruction is identical to CHK2 except that it sets condition codes rather than taking an exception when the value in R_n is out of bounds.

Condition Codes:

X	N	Z	V	C
—	U	*	U	*

X Not affected.

N Undefined.

Z Set if R_n is equal to either bound. Cleared otherwise.

V Undefined.

C Set if R_n is out of bounds. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS					
D/A		REGISTER			0	0	0	0	0	0	0	0	0	0	0
										MODE	REGISTER				

Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the location of the bounds pair. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

D/A field — Specifies whether an address register or data register is compared.

0 — Data register.

1 — Address register.

Register field — Specifies the address or data register that contains the value to be checked.

Operation: If cpcc true then scanPC + d ↗ PC

Assembler

Syntax: cpBcc <label>

Attributes: Size = (Word, Long)

Description: If the specified coprocessor condition is true, program execution continues at location scanPC + displacement. The value of the scanPC is the address of the first displacement word. The displacement is a twos complement integer that represents the relative distance in bytes from the scanPC to the destination PC. The displacement can be either 16 bits or 32 bits. The coprocessor determines the specific condition from the condition field in the operation word.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	1	SIZE	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
WORD OR															
LONG WORD DISPLACEMENT															

Instruction Fields:

- Cp-Id field — Identifies the coprocessor for this operation. Cp-Id of 000 results in an F-line exception for the MC68030.
- Size field — Specifies the size of the displacement.
 - 0 — The displacement is 16 bits.
 - 1 — The displacement is 32 bits.
- Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor which provides directives to the main processor for processing this instruction.
- 16-Bit Displacement field — The displacement value occupies 16 bits.
- 32-Bit Displacement field — The displacement value occupies 32 bits.

Operation: If cpcc false then ($D_n - 1 \nrightarrow D_n$; If $D_n \neq -1$ then $\text{scanPC} + d \nrightarrow \text{PC}$)

Assembler

Syntax: cpDBcc $D_n, \langle \text{label} \rangle$

Attributes: Size = (Word)

Description: If the specified coprocessor condition is true, execution continues with the next instruction. Otherwise, the low-order word in the specified data register is decremented by one. If the result is equal to -1 , execution continues with the next instruction. If the result is not equal to -1 , execution continues at the location indicated by the value of the scanPC plus the sign extended 16-bit displacement. The value of the scanPC is the address of the displacement word. The displacement is a twos complement integer that represents the relative distance in bytes from the scanPC to the destination PC. The coprocessor determines the specific condition from the condition word which follows the operation word.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	0	1	0	0	1	REGISTER		
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
DISPLACEMENT (16 BIT)															

Instruction Fields:

Cp-Id field — Identifies the coprocessor for this operation. Cp-Id of 000 results in an F-line exception for the MC68030.

Register field — Specifies the data register used as the counter.

Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor which provides directives to the main processor for processing this instruction.

Displacement field — Specifies the distance of the branch (in bytes).

Operation: Pass Command Word to Coprocessor

Assembler

Syntax: cpGEN <parameters as defined by coprocessor>

Attributes: Unsized

Description: Transfers the command word that follows the operation word to the specified coprocessor. The coprocessor determines the specific operation from the command word. Usually a coprocessor defines specific instances of this instruction to provide its instruction set.

Condition Codes:

May be modified by coprocessor. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
COPROCESSOR-DEPENDENT COMMAND WORD															
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSION WORDS															

Instruction Fields:

Cp-Id field — Identifies the coprocessor for this operation. Note that Cp-Id of 000 is reserved for MMU instructions for the MC68030.

Effective Address field — Specifies the location of any operand not resident in the coprocessor. The allowable addressing modes are determined by the operation to be performed.

Coprocessor Command field — Specifies the coprocessor operation to be performed. This word is passed to the coprocessor, which in turn provides directives to the main processor for processing this instruction.

Operation: If cpcc true then 1s \rightarrow Destination
else 0s \rightarrow Destination

2

Assembler

Syntax: cpScc <ea>

Attributes: Size = (Byte)

Description: Tests the specified coprocessor condition code; if the condition is true, the byte specified by the effective address is set to TRUE (all ones), otherwise that byte is set to FALSE (all zeros). The coprocessor determines the specific condition from the condition word that follows the operation word.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	0	1	EFFECTIVE ADDRESS					
											MODE		REGISTER		
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION					
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSION WORDS															

Instruction Fields:

Cp-Id field — Identifies the coprocessor for this operation. Cp-Id of 000 results in an F-line exception for the MC68030.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which in turn provides directives to the main processor for processing this instruction.

cpTRAPcc

Trap on Coprocessor Condition
(MC68020/MC68030)

cpTRAPcc

Operation: If cpcc true then TRAP

Assembler cpTRAPcc

Syntax: cpTRAPcc #⟨data⟩

Attributes: Unsized or Size = (Word, Long)

Description: Tests the specified coprocessor condition code; if the selected coprocessor condition is true, the processor initiates a cpTRAPcc exception, vector number 7. The program counter value placed on the stack is the address of the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction. The coprocessor determines the specific condition from the condition word that follows the operation word. Following the condition word is a user-defined data operand specified as immediate data to be used by the trap handler.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	0	1	1	1	1	OPMODE		
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
OPTIONAL WORD															
OR LONG WORD OPERAND															

Instruction Fields:

Cp-Id field — Identifies the coprocessor for this operation. Cp-Id of 000 results in an F-line exception for the MC68030.

Opmode field — Selects the instruction form.

010 — Instruction is followed by one operand word.

011 — Instruction is followed by two operand words.

100 — Instruction has no following operand words.

Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

Operation: If condition false then $(Dn - 1 \nrightarrow Dn;$
If $Dn \neq -1$ then $PC + d \nrightarrow PC)$

Assembler

Syntax: DBcc Dn,<label>

Attributes: Size = (Word)

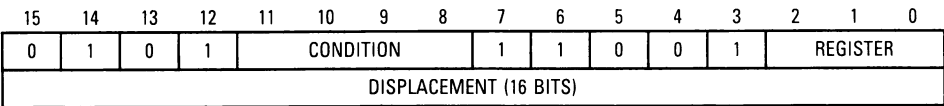
Description: Controls a loop of instructions. The parameters are: a condition code, a data register (counter), and a displacement value. The instruction first tests the condition (for termination); if it is true, no operation is performed. If the termination condition is not true, the low-order 16 bits of the counter data register are decremented by one. If the result is -1 , execution continues with the next instruction. If the result is not equal to -1 , execution continues at the location indicated by the current value of the PC plus the sign-extended 16-bit displacement. The value in the PC is the address of the instruction word of the DBcc instruction plus two. The displacement is a twos complement integer that represents the relative distance in bytes from the current PC to the destination PC.

Condition code cc specifies one of the following conditions:

CC	carry clear	0100	\overline{C}
CS	carry set	0101	C
EQ	equal	0111	Z
F	never equal	0001	0
GE	greater or equal	1100	$N \cdot V + \overline{N \cdot \overline{V}}$
GT	greater than	1110	$\overline{N \cdot V} \cdot \overline{Z} + N \cdot V \cdot \overline{Z}$
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N \cdot V}$
LS	low or same	0011	$C + Z$
LT	less than	1101	$N \cdot \overline{V} + \overline{N \cdot V}$
MI	minus	1011	\overline{N}
NE	not equal	0110	\overline{Z}
PL	plus	1010	N
T	always true	0000	1
VC	overflow clear	1000	\overline{V}
VS	overflow set	1001	V

Condition Codes:
Not affected.

Instruction Format:



Instruction Fields:

- Condition field — The binary code for one of the conditions listed in the table.
- Register field — Specifies the data register used as the counter.
- Displacement field — Specifies the number of bytes to branch.

Notes:

1. The terminating condition is similar to the UNTIL loop clauses of high-level languages. For example: DBMI can be stated as “decrement and branch until minus”.
2. Most assemblers accept DBRA for DBF for use when only a count terminates the loop (no condition is tested).
3. A program can enter a loop at the beginning or by branching to the trailing DBcc instruction. Entering the loop at the beginning is useful for indexed addressing modes and dynamically specified bit operations. In this case, the control index count must be one less than the desired number of loop executions. However, when entering a loop by branching directly to the trailing DBcc instruction, the control count should equal the loop execution count. In this case, if a zero count occurs, the DBcc instruction does not branch, and the main loop is not executed.

DIVS, DIVSL

Signed Divide
(M68000 Family)

DIVS, DIVSL

2

Operation: Destination/Source \blacktriangleright Destination

Assembler DIVS.W $\langle ea \rangle$, Dn 32/16 \blacktriangleright 16r:16q

Syntax: *DIVS.L $\langle ea \rangle$, Dq 32/32 \blacktriangleright 32q

*DIVS.L $\langle ea \rangle$, Dr:Dq 64/32 \blacktriangleright 32r:32q

*DIVSL.L $\langle ea \rangle$, Dr:Dq 32/32 \blacktriangleright 32r:32q

*Applies to MC68020/MC68030/MC68040/CPU32 only

Attributes: Size = (Word, Long)

Description: Divides the signed destination operand by the signed source operand and stores the signed result in the destination. The instruction uses one of four forms. The word form of the instruction divides a long word by a word. The result is a quotient in the lower word (least significant 16 bits) and the remainder is in the upper word (most significant 16 bits) of the result. The sign of the remainder is the same as the sign of the dividend.

The first long form divides a long word by a long word. The result is a long quotient; the remainder is discarded.

The second long form divides a quad word (in any two data registers) by a long word. The result is a long-word quotient and a long-word remainder.

The third long form divides a long word by a long word. The result is a long-word quotient and a long-word remainder.

Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before the instruction completes. If the instruction detects an overflow, it sets the overflow condition code, and the operands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X Not affected.

N Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero occurs.

DIVS, DIVSL

Signed Divide
(M68000 Family)

DIVS, DIVSL

2

- Z Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero occurs.
- V Set if division overflow occurs; undefined if divide by zero occurs. Cleared otherwise.
- C Always cleared.

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Register field — Specifies any of the eight data registers. This field always specifies the destination operand.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

Instruction Format (long word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	EFFECTIVE ADDRESS					
								MODE				REGISTER			
0	REGISTER Dq			1	SIZE	0	0	0	0	0	0	0	REGISTER Dr		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

MC68020, MC68030, MC68040, AND CPU32 ONLY

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(d ₈ ,PC,Xn)	111	011
(bd,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register Dq field — Specifies a data register for the destination operand. The low-order 32 bits of the dividend comes from this register, and the 32-bit quotient is loaded into this register.

Size field — Selects a 32- or 64-bit division operation.

0 — 32-bit dividend is in Register Dq.

1 — 64-bit dividend is in Dr:Dq.

Register Dr field — After the division, this register contains the 32-bit remainder. If Dr and Dq are the same register, only the quotient is returned. If Size is 1, this field also specifies the data register that contains the high-order 32 bits of the dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit signed integer.

Operation: Destination/Source ∇ Destination

Assembler DIVU.W <ea>,Dn 32/16 ∇ 16r:16q

Syntax: *DIVU.L <ea>,Dq 32/32 ∇ 32q

*DIVU.L <ea>,Dr:Dq 64/32 ∇ 32r:32q

*DIVUL.L <ea>,Dr:Dq 32/32 ∇ 32r:32q

*Applies to MC68020/MC68030/MC68040/CPU32 only

Attributes: Size = (Word, Long)

Description: Divides the unsigned destination operand by the unsigned source operand and stores the unsigned result in the destination. The instruction uses one of four forms. The word form of the instruction divides a long word by a word. The result is a quotient in the lower word (least significant 16 bits) and the remainder is in the upper word (most significant 16 bits) of the result.

The first long form divides a long word by a long word. The result is a long quotient; the remainder is discarded.

The second long form divides a quad word (in any two data registers) by a long word. The result is a long-word quotient and a long-word remainder.

The third long form divides a long word by a long word. The result is a long-word quotient and a long-word remainder.

Two special conditions may arise during the operation:

1. Division by zero causes a trap.
2. Overflow may be detected and set before the instruction completes. If the instruction detects an overflow, it sets the overflow condition code, and the operands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

X Not affected.

N Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero occurs.

- Z** Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero occurs.
- V** Set if division overflow occurs; undefined if divide by zero occurs. Cleared otherwise.
- C** Always cleared.

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER			0	1	1	EFFECTIVE ADDRESS MODE REGISTER					

Instruction Fields:

Register field — Specifies any of the eight data registers. This field always specifies the destination operand.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

NOTE

Overflow occurs if the quotient is larger than a 16-bit signed integer.

Instruction Format (long word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	EFFECTIVE ADDRESS					
								MODE				REGISTER			
0	REGISTER Dq			0	SIZE	0	0	0	0	0	0	0	REGISTER Dr		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

MC68020, MC68030, MC68040, AND CPU32 ONLY

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)*	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)*	111	011

MC68020, MC68030, AND MC68040 ONLY

([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register Dq field — Specifies a data register for the destination operand. The low-order 32 bits of the dividend comes from this register, and the 32-bit quotient is loaded into this register.

Size field — Selects a 32- or 64-bit division operation.

0 — 32-bit dividend is in Register Dq.

1 — 64-bit dividend is in Dr:Dq.

Register Dr field — After the division, this register contains the 32-bit remainder. If Dr and Dq are the same register, only the quotient is returned. If Size is 1, this field also specifies the data register that contains the high-order 32 bits of the dividend.

NOTE

Overflow occurs if the quotient is larger than a 32-bit unsigned integer.

EOR

Exclusive-OR Logical (M68000 Family)

EOR

2

Operation: Source \oplus Destination \rightarrow Destination

Assembler

Syntax: EOR Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Performs an exclusive-OR operation on the destination operand using the source operand and stores the result in the destination location. The size of the operation may be specified to be byte, word, or long. The source operand must be a data register. The destination operand is specified in the effective address field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

Register field — Specifies any of the eight data registers.

Opmode field:

Byte	Word	Long	Operation
100	101	110	$((\langle ea \rangle) \oplus ((\langle Dn \rangle) \rightarrow \langle ea \rangle))$

EOR

Exclusive-OR Logical (M68000 Family)

EOR

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
[(bd,An,Xn),od]	110	reg. number:An
[(bd,An),Xn,od]	110	reg. number:An

(bd,PC,Xn)*	—	—
[(bd,PC,Xn),od]	—	—
[(bd,PC),Xn,od]	—	—

*Can be used with CPU32.

NOTE

Memory-to-data register operations are not allowed. Most assemblers use EORI when the source is immediate data.

EORI

Exclusive-OR Immediate (M68000 Family)

EORI

Operation: Immediate Data \oplus Destination \blacktriangledown Destination

Assembler

Syntax: EORI #(data),<ea>

Attributes: Size = (Byte, Word, Long)

Description: Performs an exclusive-OR operation on the destination operand using the immediate data and the destination operand and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
0	0	0	0	1	0	1	0	SIZE		EFFECTIVE ADDRESS											
										MODE			REGISTER								
WORD DATA (16 BITS)								BYTE DATA (8 BITS)													
LONG DATA (32 BITS)																					

Instruction Fields:

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Immediate field — (Data immediately following the instruction):

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is next two immediate words.

EORI to CCR

Exclusive-OR Immediate
to Condition Code
(M68000 Family)

EORI to CCR

Operation: Source \oplus CCR \rightarrow CCR

Assembler

Syntax: EORI #<data>,CCR

Attributes: Size = (Byte)

Description: Performs an exclusive-OR operation on the condition code register using the immediate operand and stores the result in the condition code register (low-order byte of the status register). All implemented bits of the condition code register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Changed if bit 4 of immediate operand is one. Unchanged otherwise.
- N Changed if bit 3 of immediate operand is one. Unchanged otherwise.
- Z Changed if bit 2 of immediate operand is one. Unchanged otherwise.
- V Changed if bit 1 of immediate operand is one. Unchanged otherwise.
- C Changed if bit 0 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	BYTE DATA (8 BITS)							

EXG

Exchange Registers (M68000 Family)

EXG

2

Operation: Rx \leftrightarrow Ry

Assembler EXG Dx,Dy

Syntax: EXG Ax,Ay
EXG Dx,Ay
EXG Ay, Dx

Attributes: Size = (Long)

Description: Exchanges the contents of two 32-bit registers. The instruction performs three types of exchanges:

1. Exchange data registers.
2. Exchange address registers.
3. Exchange a data register and an address register.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER Rx				1	OPMODE				REGISTER Ry		

Instruction Fields:

Register Rx field — Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the data register.

Opmode field — Specifies the type of exchange:

01000 — Data registers.

01001 — Address registers.

10001 — Data register and address register.

Register Ry field — Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the address register.

EXT, EXTB

Sign Extend
(M68000 Family)

EXT, EXTB

2

Operation: Destination Sign-Extended \rightarrow Destination

Assembler EXT.W Dn extend byte to word
Syntax: EXT.L Dn extend word to long word
EXTB.L Dn extend byte to long word (MC68020/
MC68030/MC68040/CPU32)

Attributes: Size = (Word, Long)

Description: Extends a byte in a data register to a word or a long word, or a word in a data register to a long word, by replicating the sign bit to the left. If the operation extends a byte to a word, bit [7] of the designated data register is copied to bits [15:8] of that data register. If the operation extends a word to a long word, bit [15] of the designated data register is copied to bits [31:16] of the data register. The EXTB form copies bit [7] of the designated register to bits [31:8] of the data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	OPMODE			0	0	0	REGISTER		

Instruction Fields:

- Opmode field — Specifies the size of the sign-extension operation:
 - 010 — Sign extend low-order byte of data register to word.
 - 011 — Sign extend low-order word of data register to long.
 - 111 — Sign extend low-order byte of data register to long.
- Register field — Specifies the data register is to be sign extended.

Operation: *SSP – 2 ♦ SSP; Vector Offset ♦ (SSP);
 SSP – 4 ♦ SSP; PC ♦ (SSP);
 SSP – 2 ♦ SSP; SR ♦ (SSP);
 Illegal Instruction Vector Address ♦ PC

*The MC68000 and MC68008 cannot write the vector offset and format code to the system stack.

Assembler

Syntax: ILLEGAL

Attributes: Unsized

Description: Forces an illegal instruction exception, vector number 4. All other illegal instruction bit patterns are reserved for future extension of the instruction set and should not be used to force an exception.

Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

JMP

Jump
(M68000 Family)

JMP

2

Operation: Destination Address ♦ PC

Assembler

Syntax: JMP <ea>

Attributes: Unsized

Description: Program execution continues at the effective address specified by the instruction. The addressing mode for the effective address must be a control addressing mode.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Effective Address field — Specifies the address of the next instruction. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

JSR

Jump to Subroutine (M68000 Family)

JSR

Operation: SP ← SP - 4; PC ← (SP)
Destination Address ← PC

Assembler

Syntax: JSR <ea>

Attributes: Unsize

Description: Pushes the long-word address of the instruction immediately following the JSR instruction onto the system stack. Program execution then continues at the address specified in the instruction.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Effective Address field — Specifies the address of the next instruction. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
— (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

LEA

Load Effective Address (M68000 Family)

LEA

Operation: ⟨ea⟩ ↗ An

Assembler

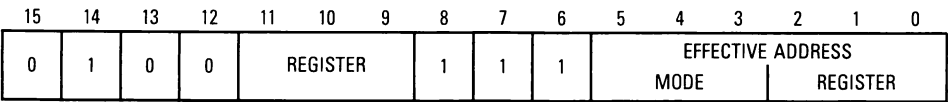
Syntax: LEA ⟨ea⟩,An

Attributes: Size = (Long)

Description: Loads the effective address into the specified address register. All 32 bits of the address register are affected by this instruction.

Condition Codes:
Not affected.

Instruction Format:



Instruction Fields:
Register field — Specifies the address register to be updated with the effective address.

Effective Address field — Specifies the address to be loaded into the address register. Only control addressing modes are allowed as shown:

2

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Operation: $Sp - 4 \nabla Sp; An \nabla (SP);$
 $SP \nabla An; SP + d \nabla SP$

Assembler

Syntax: `LINK An, #⟨displacement⟩`

Attributes: `Size = (Word, Long*)`
 `*(MC68020/MC68030/MC68040 only)`

Description: Pushes the contents of the specified address register onto the stack. Then loads the updated stack pointer into the address register. Finally, adds the displacement value to the stack pointer. For word-size operation, the displacement is the sign-extended word following the operation word. For long-size operation, the displacement is the long word following the operation word. The address register occupies one long word on the stack. The user should specify a negative displacement in order to allocate stack area.

Condition Codes:
Not affected.

Instruction Format (Word):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	REGISTER		
WORD DISPLACEMENT															

Instruction Format (Long):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	1	REGISTER		
HIGH-ORDER DISPLACEMENT															
LOW-ORDER DISPLACEMENT															

Instruction Fields:

Register field — Specifies the address register for the link.

Displacement field — Specifies the twos complement integer to be added to the stack pointer.

2

NOTE

LINK and UNLK can be used to maintain a linked list of local data and parameter areas on the stack for nested subroutine calls.

LSL,LSR

Logical Shift
(M68000 Family)

LSL,LSR

2

Operation: Destination Shifted by <count> ➔ Destination

Assembler LSd Dx,Dy

Syntax: LSd #(data),Dy
LSd <ea>
where d is direction, L or R

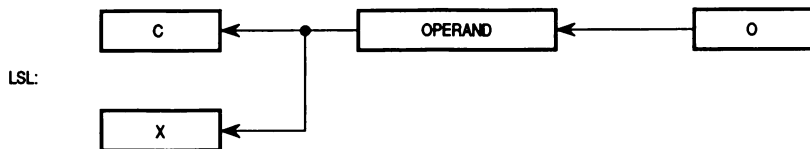
Attributes: Size = (Byte, Word, Long)

Description: Shifts the bits of the operand in the direction specified (L or R). The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register is specified in two different ways:

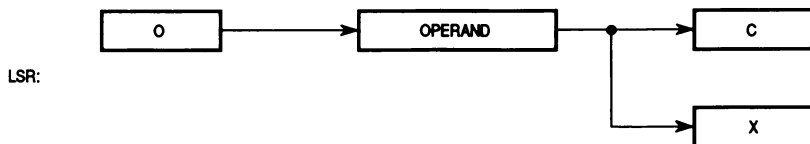
1. Immediate — The shift count (1–8) is specified in the instruction.
2. Register — The shift count is the value in the data register specified in the instruction modulo 64.

The size of the operation for register destinations may be specified as byte, word, or long. The contents of memory, <ea>, can be shifted one bit only, and the operand size is restricted to a word.

The LSL instruction shifts the operand to the left the number of positions specified as the shift count. Bits shifted out of the high-order bit go to both the carry and the extend bits; zeros are shifted into the low-order bit.



The LSR instruction shifts the operand to the right the number of positions specified as the shift count. Bits shifted out of the low-order bit go to both the carry and the extend bits; zeros are shifted into the high-order bit.



Condition Codes:

X	N	Z	V	C
*	*	*	0	*

- X Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.

Instruction Format (Register Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/REGISTER			dr	SIZE		i/r	0	1	REGISTER		

Instruction Field (Register Shifts):

Count/Register field:

If $i/r = 0$, this field contains the shift count. The values 1–7 represent shifts of 1–7; value of zero specifies a shift count of eight.

If $i/r = 1$, the data register specified in this field contains the shift count (modulo 64).

dr field — Specifies the direction of the shift:

0 — Shift right.

1 — Shift left.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

i/r field:

If $i/r = 0$, specifies immediate shift count.

If $i/r = 1$, specifies register shift count.

Register field — Specifies a data register to be shifted.

Instruction Format (Memory Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	dr	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields (Memory Shifts):

dr field — Specifies the direction of the shift:

0 — Shift right.

1 — Shift left.

Effective Address field — Specifies the operand to be shifted. Only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
{An}	010	reg. number:An
{An}+	011	reg. number:An
– {An}	100	reg. number:An
{d16,An}	101	reg. number:An
{dg,An,Xn}	110	reg. number:An

Addressing Mode	Mode	Register
{xxx}.W	111	000
{xxx}.L	111	001
#(data)	—	—
{d16,PC}	—	—
{dg,PC,Xn}	—	—

MC68020, MC68030, AND MC68040 ONLY

{bd,An,Xn}*	110	reg. number:An
{[{bd,An,Xn],od}	110	reg. number:An
{[{bd,An],Xn,od}	110	reg. number:An

{bd,PC,Xn}*	—	—
{[{bd,PC,Xn],od}	—	—
{[{bd,PC],Xn,od}	—	—

*Can be used with CPU32.

MOVE

Move Data from Source to Destination
(M68000 Family)

MOVE

Operation: Source ∇ Destination

Assembler

Syntax: MOVE <ea>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Moves the data at the source to the destination location, and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the result is negative. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0		SIZE		DESTINATION						SOURCE			
						REGISTER			MODE			MODE		REGISTER	

Instruction Fields:

Size field — Specifies the size of the operand to be moved:

01 — Byte operation.

11 — Word operation.

10 — Long operation.

MOVE

Move Data from Source to Destination (M68000 Family)

MOVE

Destination Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

MOVE

Move Data from Source to Destination (M68000 Family)

MOVE

Source Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*For byte size operation, address register direct is not allowed.
**Can be used with CPU32.

Notes:

1. Most assemblers use MOVEA when the destination is an address register.
2. MOVEQ can be used to move an immediate 8-bit value to a data register.

MOVEA

Move Address
(M68000 Family)

MOVEA

Operation: Source ↗ Destination

Assembler

Syntax: MOVEA <ea>,An

Attributes: Size = (Word, Long)

Description: Moves the contents of the source to the destination address register.
The size of the operation is specified as word or long. Word-size source operands are sign extended to 32-bit quantities.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE	DESTINATION REGISTER		0	0	1	SOURCE MODE		REGISTER					

Instruction Fields:

- Size field — Specifies the size of the operand to be moved:
- 11 — Word operation. The source operand is sign extended to a long operand and all 32 bits are loaded into the address register.
 - 10 — Long operation.
- Destination Register field — Specifies the destination address register.

MOVEA

Move Address
(M68000 Family)

MOVEA

Effective Address field — Specifies the location of the source operand. All addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

MOVE from CCR

Move from the
Condition Code Register
(MC68010/MC68020/MC68030/MC68040)

MOVE from CCR

Operation: CCR → Destination

Assembler

Syntax: MOVE CCR,<ea>

Attributes: Size = (Word)

Description: Moves the condition code bits (zero extended to word size) to the destination location. The operand size is a word. Unimplemented bits are read as zeros.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

MOVE from CCR

Move from the
Condition Code Register
(MC68010/MC68020/MC68030/MC68040)

MOVE from CCR

Instruction Fields:

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

NOTE

MOVE from CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE to CCR

Move to Condition Codes
(M68000 Family)

MOVE to CCR

2

Operation: Source \rightarrow CCR

Assembler

Syntax: MOVE (ea),CCR

Attributes: Size = (Word)

Description: Moves the low-order byte of the source operand to the condition code register. The upper byte of the source operand is ignored; the upper byte of the status register is not altered.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set to the value of bit 4 of the source operand.
- N Set to the value of bit 3 of the source operand.
- Z Set to the value of bit 2 of the source operand.
- V Set to the value of bit 1 of the source operand.
- C Set to the value of bit 0 of the source operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

MOVE to CCR

Move to Condition Codes (M68000 Family)

MOVE to CCR

Instruction Fields:

Effective Address field — Specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(d8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(d8,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

NOTE

MOVE to CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE from SR

Move from the Status Register
(MC68000/MC68008)

MOVE from SR

Operation: SR ↗ Destination

Assembler

Syntax: MOVE SR,<ea>

Attributes: Size = (Word)

Description: Moves the data in the status register to the destination location. The destination is word length. Unimplemented bits are read as zeros.

Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	EFFECTIVE ADDRESS MODE REGISTER					

Instruction Fields:

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	—	—
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d ₁₆ ,An)	101	reg. number:An	(d ₁₆ ,PC)	—	—
(d ₈ ,An,Xn)	110	reg. number:An	(d ₈ ,PC,Xn)	—	—

NOTE

Use the MOVE from CCR instruction to access only the condition codes.
Memory destination is read before it is written to.

MOVE16

Move 16 Bytes Block
(MC68040)

MOVE16

2

Operation: Source Block > Destination Block

Assembler MOVE16 (Ax) + ,(Ay) +

Syntax: MOVE16 xxx.L,(An)
MOVE16 xxx.L,(An) +
MOVE16 (An),xxx.L
MOVE16 (An) + ,xxx.L

Attributes: Size = (Line)

Description: Moves the source line to the destination line. The lines are aligned to 16-byte boundaries. Applications for this instruction include coprocessor communications, memory initialization, and fast block copy operations.

MOVE16 has two formats. The postincrement format uses the postincrement addressing mode for both source and destination, while the absolute format specifies an absolute long address for either the source or destination.

Line transfers are performed using burst reads and writes which begin with the long word pointed to by the <ea> of the source and destination, respectively. An address register used in the postincrement addressing mode is incremented by 16 after the transfer.

Example: MOVE16 (A0) + , \$FE802 A0 = \$1400F

The line at address \$14000 is read into a temporary holding register by a burst read transfer starting with long word \$1400C. Address values in A0 of \$14000–\$1400F cause the same line to be read, starting at different long words. The line is then written to the line at address \$FE800 beginning with long word \$FE800. After the instruction A0 contains \$1401F.

Source line at \$14000

\$14000	\$14004	\$14008	\$1400C
LONG WORD 0	LONG WORD 1	LONG WORD 2	LONG WORD 3

Destination line at \$FE8000

\$FE800	\$FE804	\$FE808	\$FE80C
LONG WORD 0	LONG WORD 1	LONG WORD 2	LONG WORD 3

Condition Codes:
Not affected.

Instruction Format (Postincrement source and destination):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	0	0	1	0	0	REGISTER Ax		
1	REGISTER Ay			0	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields (Postincrement source and destination):

- Register Ax — Specifies a source address register for the postincrement addressing mode.
- Register Ay — Specifies a destination address register for the postincrement addressing mode.

Instruction Format (Absolute Long Address source or destination):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	0	0	0	OPMODE	REGISTER Ay			
HIGH-ORDER ADDRESS															
LOW-ORDER ADDRESS															

MOVE16

Move 16 Bytes Block
(MC68040)

MOVE16

Instruction Fields (Absolute Long Address source and destination):

Opmode Field — Specifies the addressing modes used for source and destination:

Opmode	Source	Destination	Assembler Syntax
0 0	(Ay) +	xxx.L	MOVE16 (Ay) + ,xxx.L
0 1	xxx.L	(Ay) –	MOVE16 xxx.L.(Ay) –
1 0	(Ay)	xxx.L	MOVE16 (Ay),xxx.L
1 1	xxx.L	(Ay)	MOVE16 xxx.L,(Ay)

Register Ay — Specifies an address register for the indirect and postincrement addressing mode used as a source or destination.

32-Bit Address field — Specifies the absolute address used as a source or destination.

MOVEM

Move Multiple Registers (M68000 Family)

MOVEM

2

Operation: Registers ∇ Destination
Source ∇ Registers

Assembler MOVEM register list,(ea)

Syntax: MOVEM (ea),register list

Attributes: Size = (Word, Long)

Description: Moves the contents of selected registers to or from consecutive memory locations starting at the location specified by the effective address. A register is selected if the bit in the mask field corresponding to that register is set. The instruction size determines whether 16 or 32 bits of each register are transferred. In the case of a word transfer to either address or data registers, each word is sign extended to 32 bits, and the resulting long word is loaded into the associated register.

Selecting the addressing mode also selects the mode of operation of the MOVEM instruction, and only the control modes, the predecrement mode, and the postincrement mode are valid. If the effective address is specified by one of the control modes, the registers are transferred starting at the specified address, and the address is incremented by the operand length (2 or 4) following each transfer. The order of the registers is from data register 0 to data register 7, then from address register 0 to address register 7.

If the effective address is specified by the predecrement mode, only a register-to-memory operation is allowed. The registers are stored starting at the specified address minus the operand length (2 or 4), and the address is decremented by the operand length following each transfer. The order of storing is from address register 7 to address register 0, then from data register 7 to data register 0. When the instruction has completed, the decremented address register contains the address of the last operand stored. For the MC68020, MC68030, MC68040, and CPU32, if the addressing register is also moved to memory, the value written is the initial register value decremented by the size of the operation. The MC68000 and MC68010 write the initial register value (not decremented).

MOVEM

Move Multiple Registers (M68000 Family)

MOVEM

2

If the effective address is specified by the postincrement mode, only a memory-to-register operation is allowed. The registers are loaded starting at the specified address; the address is incremented by the operand length (2 or 4) following each transfer. The order of loading is the same as that of control mode addressing. When the instruction has completed, the incremented address register contains the address of the last operand loaded plus the operand length. If the addressing register is also loaded from memory, the memory value is ignored and the register is written with the postincremented effective address.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	SIZE	EFFECTIVE ADDRESS					
										MODE	REGISTER				
REGISTER LIST MASK															

Instruction Field:

- dr field — Specifies the direction of the transfer:
- 0 — Register to memory.
 - 1 — Memory to register.
- Size field — Specifies the size of the registers being transferred:
- 0 — Word transfer.
 - 1 — Long transfer.

Effective Address field — Specifies the memory address for the operation. For register-to-memory transfers, only control alterable addressing modes or the predecrement addressing mode are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

MOVEM

Move Multiple Registers (M68000 Family)

MOVEM

For memory-to-register transfers, only control addressing modes or the post-increment addressing mode are allowed as shown:

2

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	—	—
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Register List Mask field — Specifies the registers to be transferred. The low-order bit corresponds to the first register to be transferred; the high-order bit corresponds to the last register to be transferred. Thus, both for control modes and for the postincrement mode addresses, the mask correspondence is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

For the predecrement mode addresses, the mask correspondence is reversed:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

MOVEP

Move Peripheral Data
(M68000 Family)

MOVEP

Operation: Source ♦ Destination

Assembler MOVEP Dx,(d,Ay)

Syntax: MOVEP (d,Ay),Dx

Attributes: Size = (Word, Long)

Description: Moves data between a data register and alternate bytes within the address space starting at the location specified and incrementing by two. The high-order byte of the data register is transferred first and the low-order byte is transferred last. The memory address is specified in the address register indirect plus 16-bit displacement addressing mode. This instruction was originally designed for interfacing 8-bit peripherals on a 16-bit data bus, such as the MC68000 bus.

Example: Long transfer to/from an even address.

Byte Organization in Register

31	24 23	16 15	8 7	0
HI ORDER	MID UPPER	MID LOWER	LOW ORDER	

Byte Organization in 16-Bit Memory (Low Address at Top)

15	8 7	0
HI ORDER		
MID UPPER		
MID LOWER		
LOW ORDER		

MC68040 Byte Organization in Memory

31	24 23	16 15	8 7	0
HI ORDER		MID UPPER		
MID LOWER		LOW ORDER		

Example: Word transfer to/from (odd address).

2

Byte Organization in Register

31	24	23	16	15	8	7	0
				HI ORDER		LOW ORDER	

Byte Organization in
16-Bit Memory
(Low Address at Top)

15	8	7	0
		HI ORDER	
		LOW ORDER	

MC68040 Byte Organization in Memory

31	24	23	16	15	8	7	0
				HI ORDER			
LOW ORDER							

MOVEP

Move Peripheral Data (M68000 Family)

MOVEP

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	DATA REGISTER			OPMODE			0	0	1	ADDRESS REGISTER		
DISPLACEMENT (16 BITS)															

2

Instruction Fields:

Data Register field — Specifies the data register for the instruction.

Opmode field — Specifies the direction and size of the operation:

100 — Transfer word from memory to register.

101 — Transfer long from memory to register.

110 — Transfer word from register to memory.

111 — Transfer long from register to memory.

Address Register field — Specifies the address register which is used in the address register indirect plus displacement addressing mode.

Displacement field — Specifies the displacement used in the operand address.

MOVEQ

Move Quick
(M68000 Family)

MOVEQ

Operation: Immediate Data → Destination

Assembler

Syntax: MOVEQ #⟨data⟩,Dn

Attributes: Size = (Long)

Description: Moves a byte of immediate data to a 32-bit data register. The data in an 8-bit field within the operation word is sign extended to a long operand in the data register as it is transferred.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	REGISTER			0	DATA							

Instruction Fields:

- Register field — Specifies the data register to be loaded.
- Data field — Eight bits of data, which are sign extended to a long operand.

MULS

Signed Multiply
(M68000 Family)

MULS

2

Operation: Source * Destination \nrightarrow Destination

Assembler MULS.W <ea>,Dn 16x16 \nrightarrow 32

Syntax: *MULS.L <ea>,DI 32x32 \nrightarrow 32

*MULS.L <ea>,Dh:DI 32 x 32 \nrightarrow 64

*Applies to MC68020/MC68030/MC68040/CPU32

Attributes: Size = (Word, Long)

Description: Multiplies two signed operands yielding a signed result. This instruction has a word operand form and a long-word operand form.

In the word form, the multiplier and multiplicand are both word operands, and the result is a long-word operand. A register operand is the low-order word; the upper word of the register is ignored. All 32 bits of the product are saved in the destination data register.

In the long form, the multiplier and multiplicand are both long-word operands, and the result is either a long word or a quad word. The long-word result is the low-order 32 bits of the quad word result; the high-order 32 bits of the product are discarded.

Condition Codes:

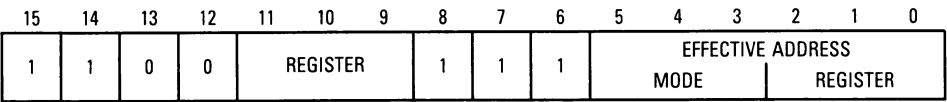
X	N	Z	V	C
—	*	*	*	0

- X Not affected.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if overflow. Cleared otherwise.
- C Always cleared.

NOTE

Overflow (V = 1) can occur only when multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if the high-order 32 bits of the quad-word product are not the sign extension of the low-order 32 bits.

Instruction Format (word form):



Instruction Fields:

Register field — Specifies a data register as the destination.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	REGISTER DI			1	SIZE	0	0	0	0	0	0	0	REGISTER Dh		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Register DI field — Specifies a data register for the destination operand. The 32-bit multiplicand comes from this register, and the low-order 32 bits of the product are loaded into this register.

Size field — Selects a 32- or 64-bit product.

0 — 32-bit product to be returned to Register DI.

1 — 64-bit product to be returned to Dh:DI.

Register Dh field — If Size is one, specifies the data register into which the high-order 32 bits of the product are loaded. If Dh = DI and Size is one, the results of the operation are undefined. Otherwise, this field is unused.

Operation: Source * Destination \rightarrow Destination

Assembler MULU.W (ea),Dn 16x16 \rightarrow 32
Syntax: *MULU.L (ea),DI 32x32 \rightarrow 32
 *MULU.L (ea),Dh:DI 32x32 \rightarrow 64
 *Applies to MC68020/MC68030/MC68040/CPU32 only.

Attributes: Size = (Word, Long)

Description: Multiplies two unsigned operands yielding an unsigned result. This instruction has a word operand form and a long-word operand form.

In the word form, the multiplier and multiplicand are both word operands, and the result is a long-word operand. A register operand is the low-order word; the upper word of the register is ignored. All 32 bits of the product are saved in the destination data register.

In the long form, the multiplier and multiplicand are both long-word operands, and the result is either a long word or a quad word. The long word result is the low-order 32 bits of the quad word result; the high-order 32 bits of the product are discarded.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

- X Not affected.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if overflow. Cleared otherwise.
- C Always cleared.

NOTE

Overflow (V=1) can occur only when multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if any of the high-order 32 bits of the quad-word product are not equal to zero.

MULU

Unsigned Multiply (M68000 Family)

MULU

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER			0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

Register field — Specifies a data register as the destination.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	REGISTER DI			0	SIZE	0	0	0	0	0	0	0	REGISTER Dh		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

Register DI field — Specifies a data register for the destination operand. The 32-bit multiplicand comes from this register, and the low-order 32 bits of the product are loaded into this register.

Size field — Selects a 32- or 64-bit product.

0 — 32-bit product to be returned to Register DI.

1 — 64-bit product to be returned to Dh:DI.

Register Dh field — If Size is one, specifies the data register into which the high-order 32 bits of the product are loaded. If Dh = DI and Size is one, the results of the operation are undefined. Otherwise, this field is unused.

Operation: $0 - (\text{Destination}_{10}) - X \nabla \text{Destination}$

Assembler

Syntax: NBCD <ea>

Attributes: Size = (Byte)

Description: Subtracts the destination operand and the extend bit from zero. The operation is performed using binary coded decimal arithmetic. The packed BCD result is saved in the destination location. This instruction produces the tens complement of the destination if the extend bit is zero, or the nines complement if the extend bit is one. This is a byte operation only.

Condition Codes:

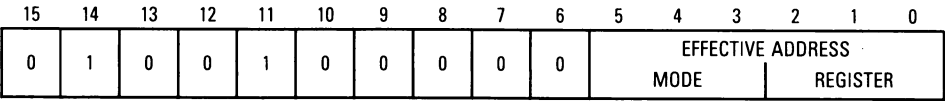
X	N	Z	V	C
*	U	*	U	*

- X Set the same as the carry bit.
- N Undefined.
- Z Cleared if the result is nonzero. Unchanged otherwise.
- V Undefined.
- C Set if a decimal borrow occurs. Cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of the operation. This allows successful tests for zero results upon completion of multiple precision operations.

Instruction Format:



Instruction Fields:

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

NEG

Negate
(M68000 Family)

NEG

2

Operation: $0 - (\text{Destination}) \rightarrow \text{Destination}$

Assembler

Syntax: NEG (ea)

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the destination operand from zero and stores the result in the destination location. The size of the operation is specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow occurs. Cleared otherwise.
- C Cleared if the result is zero. Set otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	SIZE	EFFECTIVE ADDRESS						
									MODE			REGISTER			

Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(d8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(d8,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

NEGX

Negate with Extend
(M68000 Family)

NEGX

Operation: $0 - (\text{Destination}) - X \rightarrow \text{Destination}$

Assembler

Syntax: `NEGX <ea>`

Attributes: `Size = (Byte, Word, Long)`

Description: Subtracts the destination operand and the extend bit from zero.
Stores the result in the destination location. The size of the operation is specified as byte, word, or long.

Condition Codes:

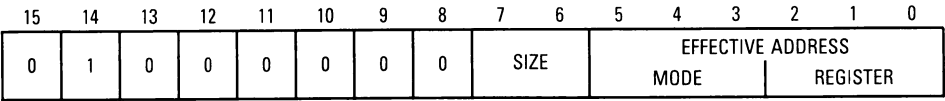
X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Cleared if the result is nonzero. Unchanged otherwise.
- V Set if an overflow occurs. Cleared otherwise.
- C Set if a borrow occurs. Cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of the operation. This allows successful tests for zero results upon completion of multiple precision operations.

Instruction Format:



Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

NOP

No Operation
(M68000 Family)

NOP

Operation: None

Assembler

Syntax: NOP

Attributes: Unsized

Description: Performs no operation. The processor state, other than the program counter, is unaffected. Execution continues with the instruction following the NOP instruction. The NOP instruction does not begin execution until all pending bus cycles are completed. This synchronizes the pipeline, and prevents instruction overlap.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT

Logical Complement (M68000 Family)

NOT

Operation: \sim Destination \blacklozenge Destination

Assembler

Syntax: NOT <ea>

Attributes: Size = (Byte, Word, Long)

Description: Calculates the ones complement of the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		

NOT

Logical Complement (M68000 Family)

NOT

Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Operation: Source V Destination \blacktriangleright Destination

Assembler OR <ea>,Dn

Syntax: OR Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Performs an inclusive-OR operation on the source operand and the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long. The contents of an address register may not be used as an operand.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the most significant bit of the result is set. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Always cleared.

C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER			OPMODE			EFFECTIVE ADDRESS					
												MODE			REGISTER

Instruction Fields:

Register field — Specifies any of the eight data registers.

Opmode field:

Byte	Word	Long	Operation
000	001	010	(<ea>) V (<Dn>) \blacktriangleright (<Dn>)
100	101	110	(<Dn>) V (<ea>) \blacktriangleright (<ea>)

OR

**Inclusive-OR Logical
(M68000 Family)**

OR

Effective Address field — If the location specified is a source operand, only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

OR

Inclusive-OR Logical
(M68000 Family)

OR

If the location specified is a destination operand, only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
((bd,An,Xn),od)	110	reg. number:An
((bd,An),Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
((bd,PC,Xn),od)	—	—
((bd,PC),Xn,od)	—	—

*Can be used with CPU32.

Notes:

1. If the destination is a data register, it must be specified using the destination Dn mode, not the destination <ea> mode.
2. Most assemblers use ORI when the source is immediate data.

ORI

Inclusive OR (M68000 Family)

ORI

2

Operation: Immediate Data V Destination ♦ Destination

Assembler

Syntax: ORI #⟨data⟩,⟨ea⟩

Attributes: Size = (Byte, Word, Long)

Description: Performs an inclusive-OR operation on the immediate data and the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

Instruction Fields:

Size field — Specifies the size of the operation.

- 00 — Byte operation.
- 01 — Word operation.
- 10 — Long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(d ₈ ,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Immediate field — (Data immediately following the instruction):

- If size = 00, the data is the low-order byte of the immediate word.
- If size = 01, the data is the entire immediate word.
- If size = 10, the data is the next two immediate words.

ORI to CCR

Inclusive-OR Immediate
to Condition Codes
(M68000 Family)

ORI to CCR

Operation: Source V CCR ∇ CCR

**Assembler
Syntax:** ORI #<data>,CCR

Attributes: Size = (Byte)

Description: Performs an inclusive-OR operation on the immediate operand and the condition codes and stores the result in the condition code register (low-order byte of the status register). All implemented bits of the condition code register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set if bit 4 of immediate operand is one. Unchanged otherwise.
- N Set if bit 3 of immediate operand is one. Unchanged otherwise.
- Z Set if bit 2 of immediate operand is one. Unchanged otherwise.
- V Set if bit 1 of immediate operand is one. Unchanged otherwise.
- C Set if bit 0 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	BYTE DATA (8 BITS)						

PACK

Pack
(MC68020/MC68030/MC68040)

PACK

2

Operation: Source (Unpacked BCD) + adjustment \blacklozenge Destination (Packed BCD)

Assembler PACK – (Ax), – (Ay), #<adjustment>

Syntax: PACK Dx,Dy,#<adjustment>

Attributes: Unsized

Description: Adjusts and packs the low four bits of each of two bytes into a single byte.

When both operands are data registers, the adjustment is added to the value contained in the source register. Bits [11:8] and [3:0] of the intermediate result are concatenated and placed in bits [7:0] of the destination register. The remainder of the destination register is unaffected.

Source (Dx):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d	x	x	x	x	e	f	g	h

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-BIT EXTENSION															

Resulting in:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x'	x'	x'	x'	a'	b'	c'	d'	x'	x'	x'	x'	e'	f'	g'	h'

Destination (Dy):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u	u	u	u	u	u	u	u	a'	b'	c'	d'	e'	f'	g'	h'

When the predecrement addressing mode is specified, two bytes from the source are fetched and concatenated. The adjustment word is added to the concatenated bytes. Bits [3:0] of each byte are extracted. These eight bits are concatenated to form a new byte which is then written to the destination.

Source (Ax):

7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d
x	x	x	x	e	f	g	h

Concatenated Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d	x	x	x	x	e	f	g	h

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-BIT EXTENSION															

Destination (Ay):

7	6	5	4	3	2	1	0
a'	b'	c'	d'	e'	f'	g'	h'

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER Dy/Ay		1	0	1	0	0	R/M	REGISTER Dx/Ax			
16-BIT EXTENSION: ADJUSTMENT															

Instruction Fields:

Register Dy/Ay field — Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

R/M field — Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field — Specifies the source register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

Adjustment field — Immediate data word that is added to the source operand.

This word is zero to pack ASCII or EBCDIC codes. Other values can be used for other codes.

PEA

Push Effective Address (M68000 Family)

PEA

Operation: $Sp - 4 \nrightarrow SP; \langle ea \rangle \nrightarrow (SP)$

Assembler

Syntax: PEA $\langle ea \rangle$

Attributes: Size = (Long)

Description: Computes the effective address and pushes it onto the stack. The effective address is a long-word address.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

Effective Address field — Specifies the address to be pushed onto the stack.
Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

PVALID

Validate a Pointer
(MC68851)

PVALID

Operation: If (source AL bits) > (destination AL bits) then Trap

Assembler PVALID VAL,<ea>

Syntax: PVALID an,<ea>

Attributes: Size = (Long)

Description: The upper bits of the source (VAL or An) are compared with the upper bits of the destination <ea>. The number of bits compared is defined by the ALC field of the AC register. If the upper bits of the source are numerically greater than (less privileged than) the destination, they cause an MMU access level exception. Otherwise, execution continues with the next instruction. If the MC field of the AC register is zero, then this instruction always causes a PMMU access level exception.

PSR: Not affected.

Instruction Format 1 (VAL contains access level to test against):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field — Specifies the logical address to be evaluated and compared against the VAL register. Only control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Instruction Format 2 (main processor register contains access level to test against):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	1	0	1	0	0	0	0	0	0	0	0	REG		

Instruction Fields:

Effective Address field — Specifies the logical address to be evaluated and compared against specified main processor address register. Only control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Note that the effective address field must provide the MC68851 with the effective address of the logical address to be validated, not the effective address describing where the PVALID operand is located. For example, in order to validate a logical address that is temporarily stored on the system stack, the instruction PVALID VAL,[(SP)] must be used since PVALID VAL,(SP) would validate the mapping on the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

Reg field — Specifies the main processor address register to be used in the compare.

ROL, ROR

Rotate (Without Extend)
(M68000 Family)

ROL, ROR

2

Operation: Destination Rotated by <count> ↗ Destination

Assembler R0d Dx,Dy

Syntax: R0d #<data>,Dy
R0d <ea>
where d is direction, L or R

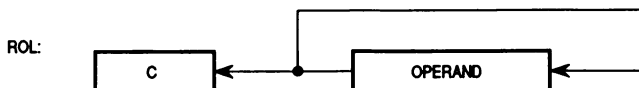
Attributes: Size = (Byte, Word, Long)

Description: Rotates the bits of the operand in the direction specified (L or R). The extend bit is not included in the rotation. The rotate count for the rotation of a register is specified in either of two ways:

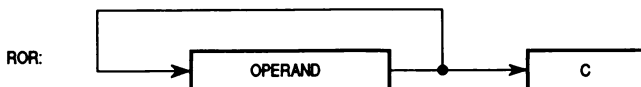
1. Immediate — The rotate count (1–8) is specified in the instruction.
2. Register — The rotate count is the value in the data register specified in the instruction, modulo 64.

The size of the operation for register destinations is specified as byte, word, or long. The contents of memory, <ea>; can be rotated one bit only, and operand size is restricted to a word.

The ROL instruction rotates the bits of the operand to the left; the rotate count determines the number of bit positions rotated. Bits rotated out of the high-order bit go to the carry bit and also back into the low-order bit.



The ROR instruction rotates the bits of the operand to the right; the rotate count determines the number of bit positions rotated. Bits rotated out of the low-order bit go to the carry bit and also back into the high-order bit.



ROL, ROR

Rotate (Without Extend)
(M68000 Family)

ROL, ROR

Condition Codes:

X	N	Z	V	C
—	*	*	0	*

- X Not affected.
N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Set according to the last bit rotated out of the operand. Cleared when the rotate count is zero.

Instruction Format (register rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/ REGISTER			dr	SIZE		i/r	1	1	REGISTER		

Instruction Fields (register rotate):

Count/Register field:

If $i/r = 0$, this field contains the rotate count. The values 1–7 represent counts of 1–7, and zero specifies a count of eight.

If $i/r = 1$, this field specifies a data register that contains the rotate count (modulo 64).

dr field — Specifies the direction of the rotate:

0 — Rotate right.

1 — Rotate left.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

i/r field — Specifies the rotate count location:

If $i/r = 0$, immediate rotate count.

If $i/r = 1$, register rotate count.

Register field — Specifies a data register to be rotated.

ROL, ROR

Rotate (Without Extend)
(M68000 Family)

ROL, ROR

Instruction Format (memory rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	dr	1	1	EFFECTIVE ADDRESS MODE REGISTER					

2

Instruction Fields (memory rotate):

dr field — Specifies the direction of the rotate:

0 — Rotate right.

1 — Rotate left.

Effective Address field — Specifies the operand to be rotated. Only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

ROXL, ROXR

Rotate with Extend
(M68000 Family)

ROXL, ROXR

Operation: Destination Rotated with X by <count> ↗ Destination

Assembler ROXd Dx,Dy

Syntax: ROXd #<data>,Dy

ROXd <ea>

where d is direction, L or R

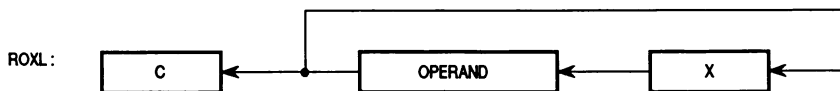
Attributes: Size = (Byte, Word, Long)

Description: Rotates the bits of the operand in the direction specified (L or R). The extend bit is included in the rotation. The rotate count for the rotation of a register is specified in either of two ways:

1. Immediate — The rotate count (1–8) is specified in the instruction.
2. Register — The rotate count is the value in the data register specified in the instruction, modulo 64.

The size of the operation for register destinations is specified as byte, word, or long. The contents of memory, <ea>, can be rotated one bit only, and operand size is restricted to a word.

The ROXL instruction rotates the bits of the operand to the left; the rotate count determines the number of bit positions rotated. Bits rotated out of the high-order bit go to the carry bit and the extend bit; the previous value of the extend bit rotates into the low-order bit.

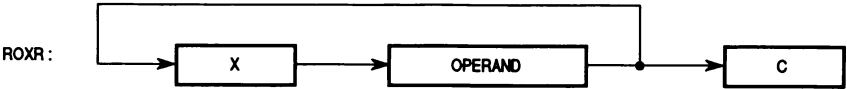


ROXL, ROXR

Rotate with Extend
(M68000 Family)

ROXL, ROXR

The ROXR instruction rotates the bits of the operand to the right; the rotate count determines the number of bit positions rotated. Bits rotated out of the low-order bit go to the carry bit and the extend bit; the previous value of the extend bit rotates into the high-order bit.



Condition Codes:

X	N	Z	V	C
*	*	*	0	*

- X Set to the value of the last bit rotated out of the operand. Unaffected when the rotate count is zero.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Set according to the last bit rotated out of the operand. When the rotate count is zero, set to the value of the extend bit.

Instruction Format (register rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/ REGISTER			dr	SIZE		i/r	1	0	REGISTER		

Instruction Fields (register rotate):

Count/Register field:

If $i/r = 0$, this field contains the rotate count. The values 1–7 represent counts of 1–7, and zero specifies a count of eight.

If $i/r = 1$, this field specifies a data register that contains the rotate count (modulo 64).

dr field — Specifies the direction of the rotate:

0 — Rotate right.

1 — Rotate left.

ROXL, ROXR

Rotate with Extend
(M68000 Family)

ROXL, ROXR

2

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

i/r field — Specifies the rotate count location:

If i/r = 0, immediate rotate count.

If i/r = 1, register rotate count.

Register field — Specifies a data register to be rotated.

Instruction Format (memory rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	dr	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields (memory rotate):

dr field — Specifies the direction of the rotate:

0 — Rotate right.

1 — Rotate left.

Effective Address field — Specifies the operand to be rotated. Only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

RTD

Return and Deallocate (MC68010/MC68020/MC68030/MC68040/CPU32)

RTD

Operation: (SP) ∇ PC; SP + 4 + d ∇ SP

Assembler

Syntax: RTD #(displacement)

Attributes: Unsized

Description: Pulls the program counter value from the stack and adds the sign-extended 16-bit displacement value to the stack pointer. The previous program counter value is lost.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0
DISPLACEMENT (16 BITS)															

Instruction Field:

Displacement field — Specifies the twos complement integer to be sign extended and added to the stack pointer.

Operation: Reload Saved Module State from Stack

Assembler

Syntax: RTM Rn

Attributes: Unsized

Description: A previously saved module state is reloaded from the top of stack. After the module state is retrieved from the top of the stack, the caller's stack pointer is incremented by the argument count value in the module state.

Condition Codes:

Set according to the content of the word on the stack.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	D/A	REGISTER		

Instruction Fields:

D/A field — Specifies whether the module data pointer is in a data or an address register.

0 — the register is a data register

1 — the register is an address register

Register field — Specifies the register number for the module data area pointer which is to be restored from the saved module state. If the register specified in A7 (SP), the updated value of the register reflects the stack pointer operations, and the saved module data area pointer is lost.

Operation: (SP) ↗ CCR; SP + 2 ↗ SP;
(SP) ↗ PC; SP + 4 ↗ SP

Assembler

Syntax: RTR

Attributes: Unsized

Description: Pulls the condition code and program counter values from the stack. The previous condition codes and program counter values are lost. The supervisor portion of the status register is unaffected.

Condition Codes:
Set to the condition codes from the stack.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS

Return from Subroutine (M68000 Family)

RTS

Operation: (SP) ∇ PC; SP + 4 ∇ SP

Assembler

Syntax: RTS

Attributes: Unsized

Description: Pulls the program counter value from the stack. The previous program counter value is lost.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD

Subtract Decimal with Extend (M68000 Family)

SBCD

2

Operation: $\text{Destination}_{10} - \text{Source}_{10} - X \rightarrow \text{Destination}$

Assembler SBCD Dx,Dy

Syntax: SBCD $-(Ax), -(Ay)$

Attributes: Size = (Byte)

Description: Subtracts the source operand and the extend bit from the destination operand and stores the result in the destination location. The subtraction is performed using binary coded decimal arithmetic; the operands are packed BCD numbers. The instruction has two modes:

1. Data register to data register: The data registers specified in the instruction contain the operands.
2. Memory to memory: The address registers specified in the instruction access the operands from memory using the predecrement addressing mode.

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

- X Set the same as the carry bit.
- N Undefined.
- Z Cleared if the result is nonzero. Unchanged otherwise.
- V Undefined.
- C Set if a borrow (decimal) is generated. Cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

SBCD

Subtract Decimal with Extend (M68000 Family)

SBCD

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER Dy/Ay			1	0	0	0	0	R/M	REGISTER Dx/Ax		

2

Instruction Fields:

Register Dy/Ay field — Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

R/M field — Specifies the operand addressing mode:

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field — Specifies the source register:

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

Operation: If Condition True
 then 1s → Destination
 else 0s → Destination

Assembler

Syntax: Scc <ea>

Attributes: Size = (Byte)

Description: Tests the specified condition code; if the condition is true, sets the byte specified by the effective address to TRUE (all ones). Otherwise, sets that byte to FALSE (all zeros). Condition code cc specifies one of the following conditions:

CC	carry clear	0100	\bar{C}
CS	carry set	0101	C
EQ	equal	0111	Z
F	never true	0001	0
GE	greater or equal	1100	$N \cdot V + \bar{N} \cdot \bar{V}$
GT	greater than	1110	$N \cdot V \cdot \bar{Z} + \bar{N} \cdot \bar{V} \cdot \bar{Z}$
HI	high	0010	$\bar{C} \cdot \bar{Z}$
LE	less or equal	1111	$Z + N \cdot \bar{V} + \bar{N} \cdot V$
LS	low or same	0011	$C + Z$
LT	less than	1101	$N \cdot \bar{V} + \bar{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	\bar{Z}
PL	plus	1010	\bar{N}
T	always true	0000	1
VC	overflow clear	1000	\bar{V}
VS	overflow set	1001	V

Condition Codes:

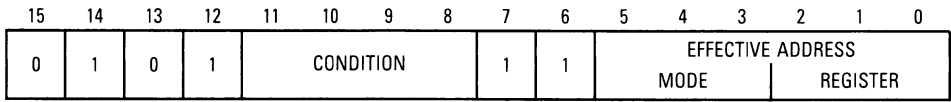
Not affected.

Scc

Set According to Condition
(M68000 Family)

Scc

Instruction Format:



Instruction Fields:

Condition field — The binary code for one of the conditions listed in the table.
Effective Address field — Specifies the location in which the true/false byte is to be stored. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

NOTE

A subsequent NEG.B instruction with the same effective address can be used to change the Scc result from TRUE or FALSE to the equivalent arithmetic value (TRUE = 1, FALSE = 0). In the MC68000 and MC68008 a memory destination is read before it is written to.

SUB

Subtract
(M68000 Family)

SUB

Operation: Destination – Source ➔ Destination

Assembler SUB <ea>,Dn
Syntax: SUB Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination operand and stores the result in the destination. The size of the operation is specified as byte, word, or long. The mode of the instruction indicates which operand is the source, which is the destination, and which is the operand size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set to the value of the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
											MODE		REGISTER		

SUB

Subtract
(M68000 Family)

SUB

Instruction Fields:

Register field — Specifies any of the eight data registers.

Opmode field:

Byte	Word	Long	Operation
000	001	010	$\langle \text{Dn} \rangle - \langle \text{ea} \rangle \rightarrow \langle \text{Dn} \rangle$
100	101	110	$\langle \text{ea} \rangle - \langle \text{Dn} \rangle \rightarrow \langle \text{ea} \rangle$

Effective Address field — Determines the addressing mode. If the location specified is a source operand, all addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*For byte size operation, address register direct is not allowed.

**Can be used with CPU32.

SUB

Subtract (M68000 Family)

SUB

If the location specified is a destination operand, only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Notes:

1. If the destination is a data register, it must be specified as a destination Dn address, not as a destination (ea) address.
2. Most assemblers use SUBA when the destination is an address register, and SUBI or SUBQ when the source is immediate data.

SUBA

Subtract Address
(M68000 Family)

SUBA

Operation: Destination – Source ➔ Destination

Assembler

Syntax: SUBA (ea),An

Attributes: Size = (Word, Long)

Description: Subtracts the source operand from the destination address register and stores the result in the address register. The size of the operation is specified as word or long. Word size source operands are sign extended to 32-bit quantities prior to the subtraction.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER				OPMODE			EFFECTIVE ADDRESS				
											MODE		REGISTER		

Instruction Fields:

Register field — Specifies the destination, any of the eight address registers.

Opmode field — Specifies the size of the operation:

011 — Word operation. The source operand is sign extended to a long operand and the operation is performed on the address register using all 32 bits.

111 — Long operation.

SUBA

Subtract Address (M68000 Family)

SUBA

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Can be used with CPU32.

SUBI

Subtract Immediate
(M68000 Family)

SUBI

Operation: Destination – Immediate Data ♦ Destination

Assembler

Syntax: SUBI #⟨data⟩,⟨ea⟩

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the immediate data from the destination operand and stores the result in the destination location. The size of the operation is specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set to the value of the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow occurs. Cleared otherwise.
- C Set if a borrow occurs. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

Instruction Fields:

Size field — Specifies the size of the operation.

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Immediate field — (Data immediately following the instruction)

If size = 00, the data is the low-order byte of the immediate word.

If size = 01, the data is the entire immediate word.

If size = 10, the data is the next two immediate words.

SUBQ

Subtract Quick
(M68000 Family)

SUBQ

Operation: Destination – Immediate Data → Destination

Assembler

Syntax: SUBQ #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the immediate data (1–8) from the destination operand. The size of the operation is specified as byte, word, or long. Only word and long operations are allowed with address registers, and the condition codes are not affected. When subtracting from address registers, the entire destination address register is used, regardless of the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set to the value of the carry bit.
- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow occurs. Cleared otherwise.
- C Set if a borrow occurs. Cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	DATA			1	SIZE		EFFECTIVE ADDRESS MODE			REGISTER		

SUBQ

Subtract Quick
(M68000 Family)

SUBQ

Instruction Fields:

Data field — Three bits of immediate data; 1–7 represent immediate values of 1–7, and zero represents eight.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the destination location. Only alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(d8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(d8,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Word and Long only.

**Can be used with CPU32.

SUBX

Subtract with Extend
(M68000 Family)

SUBX

Operation: Destination – Source – X \blacklozenge Destination

Assembler SUBX Dx,Dy

Syntax: SUBX – (Ax), – (Ay)

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand and the extend bit from the destination operand and stores the result in the destination location. The instruction has two modes:

1. Data register to data register: The data registers specified in the instruction contain the operands.
2. Memory to memory: The address registers specified in the instruction access the operands from memory using the predecrement addressing mode.

The size of the operand is specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set to the value of the carry bit.
N Set if the result is negative. Cleared otherwise.
Z Cleared if the result is nonzero. Unchanged otherwise.
V Set if an overflow occurs. Cleared otherwise.
C Set if a borrow occurs. Cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

SUBX

Subtract with Extend
(M68000 Family)

SUBX

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER Dy/Ay			1	SIZE		0	0	R/M	REGISTER Dx/Ax		

Instruction Fields:

Register Dy/Ay field — Specifies the destination register:

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

R/M field — Specifies the operand addressing mode:

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field — Specifies the source register:

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register for the predecrement addressing mode.

SWAP

Swap Register Halves
(M68000 Family)

SWAP

2

Operation: Register [31:16] \leftrightarrow Register [15:0]

Assembler

Syntax: SWAP Dn

Attributes: Size = (Word)

Description: Exchange the 16-bit words (halves) of a data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the most significant bit of the 32-bit result is set. Cleared otherwise.
- Z Set if the 32-bit result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	REGISTER		

Instruction Fields:

Register field — Specifies the data register to swap.

TAS

Test and Set an Operand (M68000 Family)

TAS

2

Operation: Destination Tested ♦ Condition Codes; 1 ♦ bit 7 of Destination

Assembler

Syntax: TAS (ea)

Attributes: Size = (Byte)

Description: Tests and sets the byte operand addressed by the effective address field. The instruction tests the current value of the operand and sets the N and Z condition bits appropriately. TAS also sets the high-order bit of the operand. The operation uses a locked or read-modify-write transfer sequence. This instruction supports use of a flag or semaphore to coordinate several processors.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the most significant bit of the operand is currently set. Cleared otherwise.
- Z Set if the operand was zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

Effective Address field — Specifies the location of the tested operand. Only data alterable addressing modes are allowed as shown:

2

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

TRAP

Trap (M68000 Family)

TRAP

Operation: 1 \blacktriangleright S Bit of SR
*SSP $- 2 \blacktriangleright$ SSP; Format/Offset \blacktriangleright (SSP);
SSP $- 4 \blacktriangleright$ SSP; PC \blacktriangleright (SSP); SSP $- 2 \blacktriangleright$ SSP;
SR \blacktriangleright (SSP); Vector Address \blacktriangleright PC
*The MC68000 and MC68008 do not write vector offset or format code to the system stack.

Assembler

Syntax: TRAP #⟨vector⟩

Attributes: Unsized

Description: Causes a TRAP #⟨vector⟩ exception. The instruction adds the immediate operand (vector) of the instruction to 32 to obtain the vector number. The range of vector values is 0–15, which provides 16 vectors.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	VECTOR			

Instruction Fields:
Vector field — Specifies the trap vector to be taken.

TRAPcc

Trap on Condition
(MC68020/MC68030/MC68040/CPU32)

TRAPcc

2

Operation: If cc then TRAP

Assembler TRAPcc

Syntax: TRAPcc.W #<data>
TRAPcc.L #<data>

Attributes: Unsized or Size = (Word, Long)

Description: If the specified condition is true, causes a TRAPcc exception. The vector number is seven. The processor pushes the address of the next instruction word (currently in the program counter) onto the stack. If the condition is not true, the processor performs no operation, and execution continues with the next instruction. The immediate data operand should be placed in the next word(s) following the operation word and is available to the trap handler. Condition code cc specifies one of the following conditions.

CC	carry clear	0100	\bar{C}
CS	carry set	0101	C
EQ	equal	0111	Z
F	never true	0001	0
GE	greater or equal	1100	$N \cdot V + \bar{N} \cdot \bar{V}$
GT	greater than	1110	$N \cdot V \cdot \bar{Z} + \bar{N} \cdot \bar{V} \cdot \bar{Z}$
HI	high	0010	$\bar{C} \cdot \bar{Z}$
LE	less or equal	1111	$Z + N \cdot \bar{V} + \bar{N} \cdot V$
LS	low or same	0011	$C + Z$
LT	less than	1100	$N \cdot \bar{V} + \bar{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	\bar{Z}
PL	plus	1010	\bar{N}
T	always true	0000	1
VC	overflow clear	1000	\bar{V}
VS	overflow set	1001	V

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	CONDITION				1	1	1	1	1	OPMODE		
OPTIONAL WORD															
OR LONG WORD															

Instruction Fields:

- Condition field — The binary code for one of the conditions listed in the table.
- Opmode field — Selects the instruction form.
- 010 — Instruction is followed by word-size operand.
 - 011 — Instruction is followed by long-word-size operand.
 - 100 — Instruction has no operand.

TRAPV

Trap on Overflow
(M68000 Family)

TRAPV

Operation: If V then TRAP

Assembler

Syntax: TRAPV

Attributes: Unsized

Description: If the overflow condition is set, causes a TRAPV exception (vector number 7). If the overflow condition is not set, the processor performs no operation and execution continues with the next instruction.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST

Test an Operand (M68000 Family)

TST

2

Operation: Destination Tested ♦ Condition Codes

Assembler

Syntax: TST (ea)

Attributes: Size = (Byte, Word, Long)

Description: Compares the operand with zero and sets the condition codes according to the results of the test. The size of the operation is specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- X Not affected.
- N Set if the operand is negative. Cleared otherwise.
- Z Set if the operand is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Fields:

Size field — Specifies the size of the operation:

00 — Byte operation.

01 — Word operation.

10 — Long operation.

Effective Address field — Specifies the addressing mode for the destination operand:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)*	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*MC68020, MC68030, MC68040, and CPU32. Address register direct allowed only for word and long.

**Can be used with CPU32.

Operation: $An \nabla SP; (SP) \nabla An; SP + 4 \nabla SP$

Assembler

Syntax: UNLK An

Attributes: Unsized

Description: Loads the stack pointer from the specified address register then loads the address register with the long word pulled from the top of the stack.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	REGISTER		

Instruction Fields:
Register field — Specifies the address register for the instruction.

UNPK

Unpack BCD
(MC68020/MC68030/MC68040)

UNPK

2

Operation: Source (Packed BCD) + adjustment → Destination (Unpacked BCD)

Assembler UNPACK – (Ax), – (Ay), #⟨adjustment⟩

Syntax: UNPK Dx,Dy,#⟨adjustment⟩

Attributes: Unsized

Description: Places the two BCD digits in the source operand byte into the lower nibbles of two bytes, and places zero bits in the upper nibbles of both bytes. Adds the adjustment value to this unpacked value. Condition codes are not altered.

When both operands are data registers, the instruction unpacks the source register contents, adds the extension word, and places the result in the destination register. The high word of the destination register is unaffected.

Source (Dx):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u	u	u	u	u	u	u	u	a	b	c	d	e	f	g	h

Intermediate Expansion:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g	h

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-BIT EXTENSION															

Destination (Dy):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
v	v	v	v	a'	b'	c'	d'	w	w	w	w	e'	f'	g'	h'

UNPK

Unpack BCD (MC68020/MC68030/MC68040)

UNPK

When the specified addressing mode is predecrement, the instruction extracts two BCD digits from a byte at the source address. After unpacking the digits and adding the adjustment word, the instruction writes the two bytes to the destination address.

Source (Ax):

7	6	5	4	3	2	1	0
a	b	c	d	e	f	g	h

Intermediate Expansion:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g	h

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-BIT EXTENSION															

Destination (Ay):

7	6	5	4	3	2	1	0
v	v	v	v	a'	b'	c'	d'
w	w	w	w	e'	f'	g'	h'

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER Dy/Ay		1	1	0	0	0	R/M	REGISTER Dx/Ax			
16-BIT EXTENSION: ADJUSTMENT															

Instruction Fields:

Register Dy/Ay field — Specifies the destination register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

R/M field — Specifies the operand addressing mode.

0 — The operation is data register to data register.

1 — The operation is memory to memory.

Register Dx/Ax field — Specifies the data register.

If R/M = 0, specifies a data register.

If R/M = 1, specifies an address register in the predecrement addressing mode.

Adjustment field — Immediate data word that is added to the source operand.

Appropriate constants can be used as the adjustment, to translate from BCD to the desired code. The constant used for ASCII is \$3030; for EBCDIC, \$F0F0.

SECTION 3

FLOATING-POINT INSTRUCTIONS

This section contains detailed information about the floating-point instructions for the MC68881, MC68882, and MC68040. Each instruction is described in detail and the instruction descriptions are arranged in alphabetical order by instruction mnemonic.

All floating-point instructions apply to the MC68881 and MC68882 processors. The MC68040 directly supports portions of the floating-point instructions through hardware, and the remainder indirectly by providing special traps and/or stack frames for the unimplemented instructions and data types. The following identification will be noted under the instruction title for the MC68040:

Directly Supported: (MC6888x/MC68040)

Software Supported: (MC6888x/040SW)

For all MC68040 floating-point instructions the “coprocessor ID” field must be 001.

Table 3-1 lists the floating-point instructions directly supported by the MC68040 and Table 3-2 lists the floating-point instructions indirectly supported.

Table 3-1. Directly Supported Floating-Point Instructions

Mnemonic	Description
FABS	Floating-Point Absolute Value
FADD	Floating-Point Add
FBcc	Floating-Point Branch Conditionally
FCMP	Floating-Point Compare
FDBcc	Floating-Point Test Condition, Decrement, and Branch
FDIV	Floating-Point Divide
FMOVE	Move Floating-Point Data Register
FMOVE	Move Floating-Point System Control Register
FMOVEM	Move Multiple Floating-Point System Data Register
FMOVEM	Move Multiple Floating-Point Control Data Register
FMUL	Floating-Point Multiply
FNEG	Floating-Point Negate
FNOP	No Operation
FRESTORE	Restore Internal Floating-Point State*
FSAVE	Save Internal Floating-Point State*
FScC	Set According to Floating-Point Condition
FSQRT	Floating-Point Square Root
FSUB	Floating-Point Subtract
FTRAPcc	Trap on Floating-Point Condition
FTST	Test Floating-Point Operand

*These are privilege instructions refer to **SECTION 4 SUPERVISOR (PRIVILEGE) INSTRUCTIONS**.

Table 3-2. Indirectly Supported Floating-Point Instructions

Mnemonic	Description
FACOS	Floating-Point Arc Cosine
FASIN	Floating-Point Arc Sine
FATAN	Floating-Point Arc Tangent
FATANH	Floating-Point Hyperbolic Arc Tangent
FCOS	Floating-Point Cosine
FCOSH	Floating-Point Hyperbolic Cosine
FETOX	Floating-Point e^x
FETOXM1	Floating-Point $e^x - 1$
FGETEXP	Floating-Point Get Exponent
FGETMAN	Floating-Point Get Mantissa
FINT	Floating-Point Integer Part
FINTRZ	Floating-Point Integer Part, Round-to-Zero
FLOG10	Floating-Point Log ₁₀
FLOG2	Floating-Point Log ₂
FLOGN	Floating-Point Log _e
FLOGNP1	Floating-Point Log _e (x + 1)
FMOD	Floating-Point Modulo Remainder
FMOVECR	Floating-Point Move Constant ROM
FREM	Floating-Point IEEE Remainder
FSCALE	Floating-Point Scale Exponent
FSGLDIV	Floating-Point Single Precision Divide
FSFLMUL	Floating-Point Single Precision Multiply
FSIN	Floating-Point Sine
FSINCOS	Floating-Point Simultaneous Sine and Cosine
FSINH	Floating-Point Hyperbolic Sine
FTAN	Floating-Point Tangent
FTANH	Floating-Point Hyperbolic Tangent
FTENTOX	Floating-Point 10^x
FTWOTOX	Floating-Point 2^x

Operation: Absolute Value of Source \rightarrow FPn

Assembler FABS.<fmt> <ea>,FPn

Syntax: FABS.X FPm,FPn

FABS.X FPn

*FrABS.<fmt> <ea>,FPn

*FrABS.X FPm,FPn

*FrABS.X FPn

where r is rounding precision, S or D

*Supported by MC68040 only.

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and stores the absolute value of that number in the destination floating-point data register.

FABS will round the result to the precision selected in the floating-point control register. FSABS and FDABS will round the result to single or double precision respectively, regardless of the rounding precision selected in the FPCR.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Absolute Value		Absolute Value		Absolute Value	

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**.

Quotient Byte: Not affected

Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NaNs
	OPERR	Cleared
	OVFL	Cleared
	UNFL	If the source is an extended precision de-normalized number, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.
	DZ	Cleared
	INEX2	Cleared
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*On if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M=0, specifies the source floating-point data register, FPM.

If R/M=1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN.

Opmode field — Specifies the instruction and rounding precision

0100010	FADD	Rounding precision specified by the FPCR.
1100010	FSADD	Single-precision rounding specified.
1100110	FDADD	Double-precision rounding specified.

Operation: Arc Cosine of Source ♦ FPn

Assembler FACOS.<fmt> <ea>,FPn

Syntax: FACOS.X FPm,FPn

FACOS.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the arc cosine of that number. Stores the result in the destination floating-point data register. This function is not defined for source operands outside of the range $[-1 \dots +1]$; if the source is not in the correct range, a NAN is returned as the result and the OPERR bit is set in the FPSR. If the source is in the correct range, the result is in the range of $[0 \dots \pi]$.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Arc Cosine		$+\pi/2$		NAN ¹	

NOTES:

- 1. Sets the OPERR bit in the FPSR exception byte.
- 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES .	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if the source is infinity, $> +1$ or < -1 ; cleared otherwise.
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for Inexact Result .

INEX1 If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	0	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID= 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M=0, specifies the source floating-point data register, FPM.

If R/M=1, specifies the source data format:

000	L	Long Word Integer
001	S	Single Precision Real
010	X	Extended Precision Real
011	P	Packed Decimal Real
100	W	Word Integer
101	D	Double Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FADD

Floating-Point Add (MC6888x/MC68040)

FADD

Operation: Source + FPn \rightarrow FPn

Assembler Syntax: FADD.<fmt> <ea>,FPn
FADD.X FPm,FPn
*FrADD.<fmt> <ea>,FPn
*FrADD.X FPm,FPn
where r is rounding precision, S or D
*Supported by MC68040 only.

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and adds that number to the number contained in the destination floating-point data register. Stores the result in the destination floating-point data register.

FADD will round the result to the precision selected in the floating-point control register. FSADD and FDADD will round the result to single or double precision respectively, regardless of the rounding precision selected in the FPCR.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	-	Add	Add	+ inf	- inf
Zero	+	-	Add	+ 0.0 0.0 ¹ 0.0 ¹ - 0.0	+ inf	- inf
Infinity	+	-	+ inf - inf	+ inf - inf	+ inf NAN ²	NAN ² - inf

- NOTES:
1. Returns +0.0 in rounding modes RN, RZ, and RP; returns -0.0 in RM.
 2. Sets the OPERR bit in the FPSR exception byte.
 3. If either operand is a NAN, refer to **1.3.4 NANs** for more information.

Floating-Point Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES .	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if the source and the destination are opposite-signed infinities; cleared otherwise.
	OVFL	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	UNFL	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	INEX1	If <fmt> is packed, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand location. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (ea) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

- 000 L Long-Word Integer
- 001 S Single-Precision Real
- 010 X Extended-Precision Real
- 011 P Packed-Decimal Real*
- 100 W Word Integer
- 101 D Double-Precision Real
- 110 B Byte Integer

*This encoding will cause an unimplemented data type exception to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN.

FADD

Floating-Point Add
(MC6888x/MC68040)

FADD

Opmode field — Specifies the instruction and rounding precision

0100010	FADD	Rounding precision specified by the FPCR.
1100010	FSADD	Single-precision rounding specified.
1100110	FDADD	Double-precision rounding specified.

FASIN

Arc Sine

(MC6888x/040SW)

FASIN

Operation: Arc Sine of the Source ♦ FPn

Assembler Syntax: FASIN.<fmt> <ea>,FPn
FASIN.X FPm,FPn
FASIN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the arc sine of the number. Stores the result in the destination floating-point data register. This function is not defined for source operands outside of the range $[-1 \dots +1]$; if the source is not in the correct range, a NAN is returned as the result and the OPERR bit is set in the FPSR. If the source is in the correct range, the result is in the range of $[-\pi/2 \dots +\pi/2]$.

3.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	Arc Sine		+0.0	–0.0	NAN ¹	

- NOTES:
1. Sets the OPERR bit in the FPSR exception byte.
 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

- Condition Codes:

Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES.**
- Quotient Byte:

Not affected
- Exception Byte:

BSUN

Cleared

SNAN

Refer to **1.3.4 NANs.**

OPERR

Set if the source is infinity, $> +1$ or < -1 ; cleared otherwise

OVFL

Cleared

UNFL

Cleared

DZ

Cleared

INEX2

Refer to appropriate user’s manual for **Inexact Result.**

INEX1 If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE				REGISTER	
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER				0	0	0	1	1	0	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FATAN

Arc Tangent
(MC6888x/040SW)

FATAN

Operation: Arc Tangent of Source \blacktriangleright FPn

Assembler FATAN.<fmt> <ea>,FPn
Syntax: FATAN.X FPm,FPn
 FATAN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the arc tangent of that number. Stores the result in the destination floating-point data register. The result is in the range of $[-\pi/2 \dots +\pi/2]$.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Arc Tangent		+ 0.0	- 0.0	+ $\pi/2$	- $\pi/2$

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**.

Quotient Byte: Not affected

Exception Byte: BSUN Cleared
 SNAN Refer to **1.3.4 NANs**.
 OPERR Cleared
 OVFL Cleared
 UNFL Refer to appropriate user's manual for **Underflow**.
 DZ Cleared
 INEX2 Refer to appropriate user's manual for **Inexact Result**.
 INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	0	1	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FATANH

Hyperbolic Arc Tangent
(MC6888x/040SW)

FATANH

Operation: Hyperbolic Arc Tangent of Source \rightarrow FPn

Assembler FATANH.<fmt> <ea>,FPn
Syntax: FATANH.X FPm,FPn
 FATANH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the hyperbolic arc tangent of that value. Stores the result in the destination floating-point data register. This function is not defined for source operands outside of the range $(-1 \dots +1)$; and the result is equal to $-$ infinity or $+$ infinity if the source is equal to $+1$ or -1 , respectively. If the source is outside of the range $[-1 \dots +1]$, a NAN is returned as the result and the OPERR bit is set in the FPSR.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Hyperbolic Arc Tangent		+0.0	-0.0	NAN ¹	

- NOTE:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if the source is $> +1$ or < -1 ; cleared otherwise
	OVFL	Cleared
	UNFL	Refer to appropriate user's manual for Underflow .

- DZ Set if the source is equal to +1 or -1; cleared otherwise
- INEX2 Refer to appropriate user's manual for **Inexact Result**.
- INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

3

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	0	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
-(An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
((bd,An,Xn),od)	110	reg. number:An
((bd,An),Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(d ₈ ,PC,Xn)	111	011
(bd,PC,Xn)	111	011
((bd,PC,Xn),od)	111	011
((bd,PC),Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M=0, specifies the source floating-point data register, FPM.

If R/M=1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is then written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: If condition true, then $PC + d \rightarrow PC$

Assembler

Syntax: FBcc.<size> <label>

Attributes: Size = (Word, Long)

Description: If the specified floating-point condition is met, program execution continues at the location $(PC) + \text{displacement}$. The displacement is a two's-complement integer that counts the relative distance in bytes. The value of the PC used to calculate the destination address is the address of the branch instruction plus two. If the displacement size is word, then a 16-bit displacement is stored in the word immediately following the instruction operation word. If the displacement size is long word, then a 32-bit displacement is stored in the two words immediately following the instruction operation word.

The conditional specifier cc selects any one of the 32 floating-point conditional tests as described in **1.3.2 Conditional Test Definitions**.

Floating-Point Status Register:

Condition Codes:	Not affected	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Set if the NAN condition code is set and the condition selected is an IEEE nonaware test
	SNAN	Not Affected
	OPERR	Not Affected
	OVFL	Not Affected
	UNFL	Not Affected
	DZ	Not Affected
	INEX2	Not Affected
	INEX1	Not Affected

Accrued Exception Byte: The IOP bit is set if the BSUN bit is set in the exception byte. No other bit is affected.

FBcc

Floating-Point Branch Conditionally

(MC6888x/MC68040)

FBcc

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	1	SIZE	CONDITIONAL PREDICATE					
16-BIT DISPLACEMENT, OR MOST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT															
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)															

Instruction Fields:

- Size field — Specifies the size of the signed displacement:

If Format=0, then the displacement is 16 bits and is sign extended before use.

If Format= 1, then the displacement is 32 bits.
- Conditional Predicate field — Specifies one of 32 conditional tests as defined in **1.3.2 Conditional Test Definitions**.

NOTE

When a BSUN exception occurs, the main processor takes a pre-instruction exception. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FBcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception occurs again immediately upon return to the routine that caused the exception.

Operation: FPN – Source

Assembler FCMP.<fmt> <ea>,FPn

Syntax: FCMP.X FPM,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and subtracts the operand from the destination floating-point data register. The result of the subtraction is not retained, but it is used to set the floating-point condition codes as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**.

Operation Table: The entries in this operation table differ from those of the tables describing most of the floating-point instructions. For each combination of input operand types, the condition code bits that may be set are indicated. If the name of a condition code bit is given and is not enclosed in brackets, then it is always set. If the name of a condition code bit is enclosed in brackets, then that bit is either set or cleared, as appropriate. If the name of a condition code bit is not given, then that bit is always cleared by the operation. The infinity bit is always cleared by the FCMP instruction, since it is not used by any of the conditional predicate equations. Note that the NAN bit is not shown, since NANs are always handled in the same manner (as described in **1.3.4 NANs**).

Destination \ Source		In Range		Zero		Infinity	
		+	–	+	–	+	–
In Range	+	{NZ}	none	none	none	N	none
	–	N	{NZ}	N	N	N	none
Zero	+	N	none	Z	Z	N	none
	–	N	none	NZ	NZ	N	none
Infinity	+	none	none	none	none	Z	none
	–	N	N	N	N	N	NZ

NOTE: If either operand is a NAN, refer to **1.3.4 NANs** for more information.

Floating-Point Status Register:

Condition Codes:
 Affected as described in the operation table above

Quotient Byte:
 Not affected

Exception Byte:
 BSUN
 Cleared
 SNAN
 Refer to **1.3.4 NANs**.
 OPERR
 Cleared
 OVFL
 Cleared
 UNFL
 Cleared
 DZ
 Cleared
 INEX2
 Cleared
 INEX1
 If <fmt> is packed, refer to appropriate user’s manual for **EXCEPTION PROCESSING**; cleared otherwise

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	1	1	0	0	0

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand location. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding in the MC68040 will cause an unimplemented data type exception to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN.

FCOS

Cosine
(MC6888x/040SW)

FCOS

Operation: Cosine of Source ♦ FPn

Assembler FCOS.<fmt> <ea>,FPn
Syntax: FCOS.X FPm,FPn
FCOS.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the cosine of that number. Stores the result in the destination floating-point data register. This function is not defined for source operands of (\pm) infinity. If the source operand is not in the range of $[-2\pi \dots +2\pi]$, then the argument is reduced to within that range before the cosine is calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) lose all accuracy. The result is in the range of $[-1 \dots +1]$.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	−	+	−	+	−
Result	Cosine		+ 1.0		NaN ¹	

- NOTE:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. If the source operand is a NaN, refer to **1.3.4 NaNs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NaNs .
	OPERR	Set if the source operand is (+ or −)infinity; cleared otherwise
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared

- INEX2

Refer to appropriate user’s manaul for **Inexact Result**.
- INEX1

If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	0	1

Instruction Fields:

- Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.
- Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should contain zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Hyperbolic Cosine of Source \rightarrow FPn

Assembler FCOSH.<fmt> <ea>,FPn

Syntax: FCOSH.X FPm,FPn
FCOSH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the hyperbolic cosine of that number. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Hyperbolic Cosine		+ 1.0		+ inf	

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte: BSUN Cleared
SNAN Refer to **1.3.4 NANs**.
OPERR Cleared
OVFL Refer to appropriate user’s manual for **Overflow**.
UNFL Cleared
DZ Cleared
INEX2 Refer to appropriate user’s manual for **Inexact Result**.
INEX1 If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	0	0	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M=0, specifies the source floating-point data register, F_{Pm}.

If R/M=1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, F_{Pn}. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: If condition true then no operation
 else $D_n - 1 \nrightarrow D_n$
 if $D_n \neq -1$
 then $PC + d \nrightarrow PC$
 else execute next instruction

Assembler

Syntax: FDBcc $D_n, <label>$

Attributes: Unsized

Description: This instruction is a looping primitive of three parameters: a floating-point condition, a counter (data register), and a 16-bit displacement. The instruction first tests the condition to determine if the termination condition for the loop has been met, and if so execution continues with the next instruction in the instruction stream. If the termination condition is not true, the low-order 16-bits of the counter register are decremented by one. If the result is -1 , the count is exhausted, and execution continues with the next instruction. If the result is not equal to -1 , execution continues at the location specified by the current value of the PC plus the sign-extended 16-bit displacement. The value of the PC used in the branch address calculation is the address of the displacement word.

The conditional specifier cc selects any one of the 32 floating-point conditional tests as described in **1.3.2 Conditional Test Definitions**.

Floating-Point Status Register:

Condition Codes:	Not affected	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Set if the NAN condition code is set and the condition selected is an IEEE nonaware test
	SNAN	Not Affected
	OPERR	Not Affected
	OVFL	Not Affected

UNFL Not Affected
DZ Not Affected
INEX2 Not Affected
INEX1 Not Affected

Accrued Exception Byte: The IOP bit is set if the BSUN bit is set in the exception byte. No other bit is affected.

Instruction Format:

3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	1	0	0	1	COUNT REGISTER		
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICATE					
16-BIT DISPLACEMENT															

Instruction Fields:

- Count Register field — Specifies data register that is used as the counter.
- Conditional Predicate field — Specifies one of the 32 floating-point conditional tests as described in **1.3.2 Conditional Test Definitions**.
- Displacement field — Specifies the branch distance (from the address of the instruction plus two) to the destination in bytes.

Notes:

1. The terminating condition is like that defined by the UNTIL loop constructs of high-level languages. For example: FDBOLT can be stated as “decrement and branch until ordered less than”.
2. There are two basic ways of entering a loop: at the beginning, or by branching to the trailing FDBcc instruction. If a loop structure terminated with FDBcc is entered at the beginning, the control counter must be one less than the number of loop executions desired. This count is useful for indexed addressing modes and dynamically specified bit operations. However, when entering a loop by branching directly to the trailing FDBcc instruction, the count should equal the loop execution count. In this case, if the counter is zero when the loop is entered, the FDBcc instruction does not branch, causing a complete bypass of the main loop.

3. When a BSUN exception occurs, a pre-instruction exception is taken by the main processor. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FDBcc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception occurs again immediately upon return to the routine that caused the exception.

Operation: $FPn \div \text{Source} \rightarrow FPn$

Assembler Syntax: `FDIV.<fmt> <ea>,FPn`
`FDIV.X FPm,FPn`
`*FrDIV.<fmt> <ea>,FPn`
`*FrDIV.X FPm,FPn`
 where r is rounding precision, S or D
 *Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and divides that number into the number in the destination floating-point data register. Stores the result in the destination floating-point data register.

FDIV will round the result to the precision selected in the floating-point control register. FSDIV and FDDIV will round the result to single or double precision respectively, regardless of the rounding precision selected in the FPCR.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	-	Divide	Divide	Divide	Divide
Zero	+	-	+0.0	+0.0	NAN ²	NAN ²
Infinity	+	-	+inf	-inf	+inf	-inf

NOTES:

1. Sets the DZ bit in the FPSR exception byte.
2. Sets the OPERR bit in the FPSR exception byte.
3. If either operand is a NAN, refer to **1.3.4 NANs** for more information.

Floating-Point Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES		
Quotient Byte:	Not affected		
Exception Byte:	BSUN	Cleared	
	SNAN	Refer to 1.3.4 NaNs .	
	OPERR	Set for 0(÷)0 or infinity(÷)infinity; cleared otherwise	
	OVFL	Refer to appropriate user's manual for EXCEPTION PROCESSING .	
	UNFL	Refer to appropriate user's manual for EXCEPTION PROCESSING .	
	DZ	Set if the source is zero and the destination is in range; cleared otherwise	
	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING .	
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.	

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand location. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding in the MC68040 will cause an unimplemented data type exception to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN.

Opmode field — Specifies the instruction and rounding precision.

0100000	FDIV	Rounding precision specified by the FPCR.
1100000	FSDIV	Single-precision rounding specified.
1100100	FDDIV	Double-precision rounding specified.

Operation: $e(\text{Source}) \rightarrow \text{FPn}$

Assembler FETOX.<fmt> <ea>,FPn

Syntax: FETOX.X FPm,FPn
 FETOX.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates e to the power of that number. Stores the result in the destination floating-point data register.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	e^x		+ 1.0		+ inf	+ 0.0

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Cleared
	OVFL	Refer to appropriate user's manual for Overflow .
	UNFL	Refer to appropriate user's manual for Underflow .
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for Inexact Result .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	0	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
[(bd,An,Xn),od)	110	reg. number:An
[(bd,An),Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
[(bd,PC,Xn),od)	111	011
[(bd,PC),Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier Field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FGETEXP

Get Exponent
(MC6888x/040SW)

FGETEXP

Operation: Exponent of Source → FPn

Assembler Syntax: FGETEXP.<fmt> <ea>,FPn
FGETEXP.X FPm,FPn
FGETEXP.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and extracts the binary exponent. Removes the exponent bias, converts the exponent to an extended precision floating-point number, and stores the result in the destination floating-point data register.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	Exponent		+ 0.0	– 0.0	NaN ¹	

- NOTES:
1. Sets the OPERR bit in the FPSR exception byte.
 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NANs .
OPERR	Set if the source is (+ or –)infinity; cleared otherwise
OVFL	Cleared
UNFL	Cleared
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	1	0

3

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(d ₈ ,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FGETMAN

Get Mantissa
(MC6888x/040SW)

FGETMAN

Operation: Mantissa of Source → FPn

Assembler FGETMAN.<fmt> <ea>,FPn
Syntax: FGETMAN.X FPm,FPn
FGETMAN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and extracts the mantissa. Converts the mantissa to an extended precision value and stores the result in the destination floating-point data register. The result is in the range [1.0 . . . 2.0) with the sign of the source mantissa, zero, or is a NAN.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	Mantissa		+0.0	–0.0	NAN ¹	

NOTES:

1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if the source is (+ or –)infinity; cleared otherwise
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Cleared
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	1	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M=0, this field is unused, and should be all zeros.

If R/M=1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Integer Part of Source \blacktriangleright FPn

Assembler FINT.<fmt> <ea>,FPn
Syntax: FINT.X FPm,FPn
FINT.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and extracts the integer part and converts it to an extended precision floating-point number. Stores the result in the destination floating-point data register. The integer part is extracted by rounding the extended precision number to an integer using the current rounding mode selected in the FPCR mode control byte. Thus, the integer part returned is the number that is to the left of the radix point when the exponent is zero, after rounding. For example, the integer part of 137.57 is 137.0 for the round-to-zero and round-to-minus infinity modes, and 138.0 for the round-to-nearest and round-to-plus infinity modes. Note that the result of this operation is a floating-point number.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Integer		+ 0.0	- 0.0	+ inf	- inf

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte: BSUN Cleared
SNAN Refer to **1.3.4 NANs**.
OPERR Cleared
OVFL Cleared
UNFL Cleared
DZ Cleared
INEX2 Refer to appropriate user's manual for **Inexact Result**.

INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	0	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

- 0 — The operation is register to register.
- 1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

- 000 L Long-Word Integer
- 001 S Single-Precision Real
- 010 X Extended-Precision Real
- 011 P Packed-Decimal Real
- 100 W Word Integer
- 101 D Double-Precision Real
- 110 B Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Integer Part of Source ♦ FPn

Assembler FINTRZ.<fmt> <ea>,FPn
Syntax: FINTRZ.X FPm,FPn
FINTRZ.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and extracts the integer part and converts it to an extended precision floating-point number. Stores the result in the destination floating-point data register. The integer part is extracted by rounding the extended precision number to an integer using the round-to-zero mode, regardless of the rounding mode selected in the FPCR mode control byte (making it useful for FORTRAN assignments). Thus, the integer part returned is the number that is to the left of the radix point when the exponent is zero. For example, the integer part of 137.57 is 137.0; the integer part of 0.1245×10^2 is 12.0. Note that the result of this operation is a floating-point number.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Integer, Forced Round-To-Zero		+ 0.0	- 0.0	+ inf	- inf

NOTE: If the source operand is a NAN, refer to 1.3.4 NANs for more information.

Status Register:

Condition Codes: Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES

Quotient Byte: Not affected

Exception Byte: BSUN Cleared
SNAN Refer to 1.3.4 NANs.
OPERR Cleared
OVFL Cleared
UNFL Cleared
DZ Cleared

INEX2 Refer to appropriate user's manual for **Inexact Result**.

INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODEREGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	1	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If RM = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Log₁₀ of Source ➔ FPn

Assembler FLOG10.<fmt> <ea>,FPn
Syntax: FLOG10.X FPm,FPn
 FLOG10.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Convert the source operand to extended precision (if necessary) and calculates the logarithm of that number using base 10 arithmetic. Stores the result in the destination floating-point data register. This function is not defined for input values less than zero.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	Log ₁₀	NAN ¹	– inf ²		+ inf	NAN ¹

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. Sets the DZ bit in the FPSR exception byte.
 - 3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

- Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**
- Quotient Byte: Not affected
- Exception Byte: BSUN Cleared
 SNAN Refer to **1.3.4 NANs**.
 OPERR Set if the source operand is <0; cleared otherwise
 OVFL Cleared
 UNFL Cleared
 DZ Set if the source is (+ or –); cleared otherwise
 INEX2 Refer to appropriate user’s manual for **Inexact Result**.

INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
										MODE		REGISTER			
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER		0	0	1	0	1	0	1		

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

- 0 — The operation is register to register.
- 1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

- If R/M = 0, specifies the source floating-point data register, FPM.
- If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Log2 of Source ♦ FPN

Assembler FLOG2.<fmt> <ea>,FPn
Syntax: FLOG2.X FPM,FPn
FLOG2.X FPN

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the logarithm of that number using base two arithmetic. Stores the result in the destination floating-point data register. This function is not defined for input values less than zero.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Log2	NAN ¹	- inf ²		+ inf	NAN ¹

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. Sets the DZ bit in the FPSR exception byte.
 - 3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NANs .
OPERR	Set if the source is < 0; cleared otherwise
OVFL	Cleared
UNFL	Cleared
DZ	Set if the source is (+ or -)0; cleared otherwise
INEX2	Refer to appropriate user's manual for Inexact Result .

INEX1 If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	1	1	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID=1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FLOGN

Log_e
(MC6888x/040SW)

FLOGN

Operation: Log_e of Source → FPn

Assembler FLOGN.<fmt> <ea>,FPn
Syntax: FLOGN.X FPm,FPn
FLOGN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the natural logarithm of that number. Stores the result in the destination floating-point data register. This function is not defined for input values less than zero.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	ln(x)	NAN ¹	– inf ²		+ inf	NAN ¹

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. Sets the DZ bit in the FPSR exception byte.
 - 3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

- Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**
- Quotient Byte: Not affected
- Exception Byte:
- BSUN Cleared
 - SNAN Refer to **1.3.4 NANs**.
 - OPERR Set if the source operand is < 0; cleared otherwise
 - OVFL Cleared
 - UNFL Cleared
 - DZ Set if the source is (+ or –)0; cleared otherwise
 - INEX2 Refer to appropriate user’s manual for **Inexact Result**.

INEX1 If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	1	0	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M=0, specifies the source floating-point data register, FPM.

If R/M=1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FLOGNP1

$\text{Log}_e (x + 1)$
(MC6888x/040SW)

FLOGNP1

Operation: Log_e of (Source + 1) ∇ FPN

Assembler FLOGNP1.<fmt> <ea>,FPn
Syntax: FLOGNP1.X FPM,FPn
FLOGNP1.X FPN

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary), adds one to that value, and calculates the natural logarithm of that intermediate result. Stores the result in the destination floating-point data register. This function is not defined for input values less than -1 .

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	$\ln(x + 1)$	$\ln(x + 1)^1$	+0.0	-0.0	+inf	NAN ²

NOTES:

1. If the source is -1 , sets the DZ bit in the FPSR exception byte and returns a NAN.
If the source is < -1 , sets the OPERR bit in the FPSR exception byte and returns a NAN.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NANs .
OPERR	Set if the source operand is < -1 ; cleared otherwise
OVFL	Cleared
UNFL	Refer to appropriate user's manual for Underflow .
DZ	Set if the source operand is -1 ; cleared otherwise

- INEX2
- Refer to appropriate user’s manual for **Inexact Result**.
- INEX1
- If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	1	1	0

Instruction Fields:

- Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.
- Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
[(bd,An,Xn),od]	110	reg. number:An
[(bd,An),Xn,od]	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
[(bd,PC,Xn),od]	111	011
[(bd,PC),Xn,od]	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Modulo Remainder of (FPn (÷) Source) ♦ FPn

Assembler Syntax: FMOD.<fmt> <ea>,FPn
FMOD.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the modulo remainder of the number in the destination floating-point data register, using the source operand as the modulus. Stores the result in the destination floating-point data register, and stores the sign and seven least significant bits of the quotient in the FPSR quotient byte (the quotient is the result of FPn (÷) Source). The modulo remainder function is defined as:

$$FPn - (Source \times N)$$

where:
 $N = INT(FPn (/) Source)$ in the round-to-zero mode

The FMOD function is not defined for a source operand equal to zero or for a destination operand equal to infinity. Note that this function is not the same as the FREM instruction, which uses the round-to-nearest mode and thus returns the remainder that is required by the *IEEE Specification for Binary Floating-Point Arithmetic*.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	Modulo Remainder	NaN ¹		FPn ²	
Zero	+	+0.0	NaN ¹		+0.0	
	-	-0.0			-0.0	
Infinity	+	NaN ¹	NaN ¹		NaN ¹	
	-					

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. Returns the value of FPn before the operation. However, the result is processed by the normal instruction termination procedure to round it as required. Thus, an overflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was loaded.
 - 3. If the source operand is a NaN, refer to 1.3.4 NaNs for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Loaded with the sign and least significant seven bits of the quotient (FPn (÷) Source). The sign of the quotient is the exclusive OR of the sign bits of the source and destination operands.	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if the source is zero, or the destination is infinity; cleared otherwise
	OVFL	Cleared
	UNFL	Refer to appropriate user's manual for Underflow .
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for Inexact Result .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE	REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	0	0	1

Instruction Fields:

- Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.
- Effective Address field — Determines the addressing mode for external operands. If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(d8,An,Xn)	110	reg. number:An	(d8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, F_{Pm}.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, F_{Pn}.

Operation: Source ∇ Destination

Assembler Syntax: FMOVE.<fmt> <ea>,FPn
 FMOVE.<fmt> FPm,<ea>
 FMOVE.P FPm,<ea>{Dn}
 FMOVE.P FPm,<ea>{#k}
 *FrMOVE.<fmt> <ea>,FPn
 where r is rounding precision, S or D
 *Supported by MC68040 only

3

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Moves the contents of the source operand to the destination operand. Although the primary function of this instruction is data movement, it is also considered an arithmetic instruction since conversions from the source operand format to the destination operand format are performed implicitly during the move operation. Also, the source operand is rounded according to the selected rounding precision and mode.

Unlike the integer data movement instruction, the floating-point move instruction does not support a memory-to-memory format (for such transfers, it is much faster to utilize the integer MOVE instruction to transfer the floating-point data than to use the FMOVE instruction). The FMOVE instruction only supports memory-to-register, register-to-register, and register-to-memory operations (in this context, memory may refer to an integer data register if the data format is byte, word, long, or single). The memory-to-register and register-to-register operations use a command word encoding distinctly different from that used by the register-to-memory operation, and these two operation classes are described separately below.

Memory-to-Register or Register-to-Register Operation:

Converts the source operand to an extended precision floating-point number (if necessary) and stores it in the destination floating-point data register. MOVE will round the result to the precision selected in the floating-point control register. FMOVE and FDMOVE will round the result to single or double precision respectively, regardless of the rounding precision selected in the FPCR. Depending on the source data format and the rounding precision, some operations

may produce an inexact result. In the following table, combinations that can produce an inexact result are marked with a dot (●), but all other combinations produce an exact result.

	Source Format:	B	W	L	S	D	X	P
Rounding Precision:	Single			●		●	●	●
	Double						●	●
	Extended							●

Floating-Point Status Register (<ea> to register):

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs.
	OPERR	Cleared
	OVFL	Cleared
	UNFL	Refer to appropriate user’s manual for EXCEPTION PROCESSING if the source is an extended-precision denormalized number; cleared otherwise.
	DZ	Cleared
	INEX2	Refer to appropriate user’s manual for EXCEPTION PROCESSING if <fmt> is L, D, or X; cleared otherwise.
	INEX1	Refer to appropriate user’s manual for EXCEPTION PROCESSING if <fmt> is P; cleared otherwise.
Accrued Exception Byte:	Affected as described in EXCEPTION PROCESSING, refer to appropriate user’s manual.	

Instruction Format (<ea> to register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODEREGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields (<ea> to register):

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, Fp_m.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding in the MC68040 will cause an unimplemented data type exception to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN.

Opmode field — Specifies the instruction and rounding precision.

0000000	FMOVE	Rounding precision specified by the FPCR.
1000000	FSMOVE	Single-precision rounding specified.
1000100	FDMOVE	Double-precision rounding specified.

Register-to-Memory Operation:

Rounds the source operand to the size of the specified destination format and stores it at the destination effective address. If the format of the destination is packed decimal, a third operand is required to specify the format of the resultant string. This operand, called the k factor, is a 7-bit signed integer (twos complement) and may be specified as an immediate value or in an integer data register. If a data register contains the k factor, only the least significant seven bits are used, and the rest of the register is ignored.

Floating-Point Status Register (register to memory):

Condition Codes:	Not affected	
Quotient Byte:	Not affected	
Exception Byte: <fmt> is B, W, or L	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if the source operand is infinity, or if the destination size is exceeded after conversion and rounding; cleared otherwise
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared

3

	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING.
	INEX1	Cleared
<fmt> is S, D, or X	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs.
	OPERR	Cleared
	OVFL	Refer to appropriate user's manual for EXCEPTION PROCESSING.
	UNFL	Refer to appropriate user's manual for EXCEPTION PROCESSING.
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING.
	INEX1	Cleared
<fmt> is P	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs.
	OPERR	Set if the k factor > +17, or the magnitude of the decimal exponent exceeds three digits; cleared otherwise
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING.
	INEX1	Cleared

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user's manual.

Instruction Format (register to memory):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	1	1	DESTINATION FORMAT			SOURCE REGISTER			K FACTOR (IF REQUIRED)						

FMOVE

Move Floating-Point Data Register
(MC6888x/MC68040)

FMOVE

Instruction Fields (register to memory):

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	—	—
(An) +	011	reg. number:An			
–(An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	—	—
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

*Only if (fmt) is Byte, Word, Long, or Single.

Destination Format field — Specifies the data format of the destination operand:

- 000

L

Long-Word Integer
- 001

S

Single-Precision Real
- 010

X

Extended-Precision Real
- 011

P{#k}

Packed-Decimal Real with Static k Factor*
- 100

W

Word Integer
- 101

D

Double-Precision Real
- 110

B

Byte Integer
- 111

P{Dn}

Packed-Decimal Real with Dynamic k Factor*
- *This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

- Source Register field — Specifies the source floating-point data register, FPM.
- k-Factor field — Only used if the destination format is packed decimal, to specify the format of the decimal string. For any other destination format, this field should be set to all zeros. For a static k-factor, this field is encoded with a twos-complement integer where the value defines the format as follows:
- 64 to 0 — Indicates the number of significant digit to the right of the decimal point (Fortran “F” format).
 - + 1 to + 17 — Indicates the number of significant digits in the mantissa (Fortran “E” format).
 - + 18 to + 63 — Sets the OPERR bit in the FPSR exception byte, treated as + 17.

The format of this field for a dynamic k-factor is:

r r r 0 0 0 0

where:

“rrr” is the number of the main processor data register that contains the k-factor value.

The following table gives several examples of how the k-factor value affects the format of the decimal string that is produced by the FPCP. The format of the string that is generated is independent of the source of the k-factor (static or dynamic).

k-Factor	Source Operand Value	Destination String
– 5	+ 12345.678765	+ 1.2345555567877 E + 4
– 3	+ 12345.678765	+ 1.2345679 E + 4
– 1	+ 12345.678765	+ 1.23457 E + 4
0	+ 12345.678765	+ 1.2346 E + 4
+ 1	+ 12345.678765	+ 1. E + 4
+ 3	+ 12345.678765	+ 1.23 E + 4
+ 5	+ 12345.678765	+ 1.2346 E + 4

FMOVE

Move Floating-Point System Control Register

(MC6888x/MC68040)

FMOVE

Operation: Source ↗ Destination

Assembler FMOVE.L <ea>,FPcr
Syntax: FMOVE.L FPcr,<ea>

Attributes: Size = (Long)

Description: Moves the contents of a floating-point system control register (FPCR, FPSR, or FPAIR) to or from an effective address. A 32-bit transfer is always performed, even though the system control register may not have 32 implemented bits. Unimplemented bits of a control register are read as zeros and are ignored during writes (but must be zero for compatability with future devices).

This instruction does not cause pending exceptions (other than protocol violations) to be reported. Furthermore, a write to the FPCR exception enable byte or the FPSR exception status byte cannot generate a new exception, regardless of the value written.

Floating-Point Status Register: Changed only if the destination is the FPSR; in which case all bits are modified to reflect the value of the source operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
1	0	dr	REGISTER SELECT			0	0	0	0	0	0	0	0	0	0

Instruction Fields:
Effective Address field — Determines the addressing mode for the operation as shown below.

FMOVE Move Floating-Point System Control Register FMOVE

(MC6888x/MC68040)

Memory-to-Register —

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if the source register is the FPIAR.

Register-to-Memory — Only alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Only if the destination register is the FPIAR.

FMOVE

Move Floating-Point System Control Register
(MC6888x/MC68040)

FMOVE

- dr field — Specifies the direction of the data transfer.
- 0 — From <ea> to the specified system control register.
 - 1 — From the specified system control register to <ea>.
- Register Select field — Specifies the system control register to be moved:
- | | | |
|-----|-------|---|
| 100 | FPCR | Floating-Point Control Register |
| 010 | FPSR | Floating-Point Status Register |
| 001 | FPIAR | Floating-Point Instruction Address Register |

FMOVECR

Move Constant ROM
(MC6888x/040SW)

FMOVECR

Operation: ROM Constant \rightarrow FPN

Assembler

Syntax: FMOVECR.X #ccc,FPn

Attributes: Format = (Extended)

Description: Fetches an extended precision constant from the FPCP on-chip ROM, rounds it to the precision specified in the FPCR mode control byte, and stores it in the destination floating-point data register. The constant is specified by a predefined offset into the constant ROM. The values of the constants contained in the ROM are shown in the offset table at the end of this description.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES													
Quotient Byte:	Not affected													
Exception Byte:	BSUN	Cleared												
	SNAN	Cleared												
	OPERR	Cleared												
	OVFL	Cleared												
	UNFL	Cleared												
	DZ	Cleared												
	INEX2	Refer to appropriate user's manual for Inexact Result.												
	INEX1	Cleared												

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	DESTINATION REGISTER			ROM OFFSET						

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Destination Register field — Specifies the destination floating-point data register, FPn.

ROM Offset field — Specifies the offset into the FPCP on-chip constant ROM where the desired constant is located. The offsets for the available constants are:

Offset	Constant
\$00	π
\$0B	$\text{Log}_{10}(2)$
\$0C	e
\$0D	$\text{Log}_2(e)$
\$0E	$\text{Log}_{10}(e)$
\$0F	0.0
\$30	$1_n(2)$
\$31	$1_n(10)$
\$32	10^0
\$33	10^1
\$34	10^2
\$35	10^4
\$36	10^8
\$37	10^{16}
\$38	10^{32}
\$39	10^{64}
\$3A	10^{128}
\$3B	10^{256}
\$3C	10^{512}
\$3D	10^{1024}
\$3E	10^{2048}
\$3F	10^{4096}

The on-chip ROM contains other constants useful only to the on-chip microcode routines. The values contained at offsets other than those defined above are reserved for the use of Motorola, and may be different on various mask sets of the FPCP.

FMOVEM

Move Multiple Floating-Point
Data Registers
(MC6888x/MC68040)

FMOVEM

Operation: Register List ∇ Destination
Source ∇ Register List

Assembler FMOVEM.X $\langle \text{list} \rangle, \langle \text{ea} \rangle$
Syntax: FMOVEM.X Dn, $\langle \text{ea} \rangle$
FMOVEM.X $\langle \text{ea} \rangle, \langle \text{list} \rangle$
FMOVEM.X $\langle \text{ea} \rangle, \text{Dn}$

$\langle \text{list} \rangle$

A list of any combination of the eight floating-point data registers, with individual register names separated by a slash (/); and/or contiguous blocks of registers specified by the first and last register names separated by a dash (–).

Attributes: Format = (Extended)

Description: Moves one or more extended precision numbers to or from a list of floating-point data registers. No conversion or rounding is performed during this operation, and the FPSR is not affected by the instruction. This instruction does not report pending exceptions.

Any combination of the eight floating-point data registers can be transferred, with the selected registers specified by a user-supplied mask. This mask is an 8-bit number, where each bit corresponds to one register; if a bit is set in the mask, that register is moved. The register select mask may be specified as a static value contained in the instruction, or a dynamic value in the least significant eight bits of an integer data register (the remaining bits of the register are ignored).

FMOVEM allows three types of addressing modes: the control modes, the predecrement mode, or the postincrement mode. If the effective address is one of the control addressing modes, the registers are transferred between the processor and memory starting at the specified address and up through higher addresses. The order of the transfer is from FP0–FP7.

FMOVEM

Move Multiple Floating-Point
 Data Registers
 (MC6888x/MC68040)

FMOVEM

If the effective address is the predecrement mode, only a register-to-memory operation is allowed. The registers are stored starting at the address contained in the address register and down through lower addresses. Before each register is stored, the address register is decremented by 12 (the size of an extended precision number in memory) and the floating-point data register is then stored at the resultant address. When the operation is complete, the address register points to the image of the last floating-point data register stored. The order of the transfer is from FP0–FP7.

If the effective address is the postincrement mode, only a memory-to-register operation is allowed. The registers are loaded starting at the specified address and up through higher addresses. After each register is stored, the address register is incremented by 12 (the size of an extended precision number in memory). When the operation is complete, the address register points to the byte immediately following the image of the last floating-point data register loaded. The order of the transfer is the same as for the control addressing modes: FP0–FP7.

Floating-Point Status Register: Not Affected. Note that the FMOVEM instruction provides the only mechanism for moving a floating-point data item between the FPU and memory without performing any data conversions or affecting the condition code and exception status bits.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
1	1	dr	MODE		0	0	0	REGISTER LIST							

FMOVEM

Move Multiple Floating-Point Data Registers (MC6888x/MC68040)

FMOVEM

Instruction Fields:

Effective Address field — Determines the addressing mode for the operation as shown below.

Memory-to-Register — Only control addressing modes or the postincrement addressing mode are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register-to-Memory — Only control alterable addressing modes or the predecrement addressing mode are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

FMOVEM

Move Multiple Floating-Point
 Data Registers
 (MC6888x/MC68040)

FMOVEM

dr field — Specifies the direction of the transfer.

- 0 — Move the listed registers from memory to the FPU.
- 1 — Move the listed registers from the FPU to memory.

Mode field — Specifies the type of the register list and addressing mode.

- 00 — Static register list, predecrement addressing mode.
- 01 — Dynamic register list, predecrement addressing mode.
- 10 — Static register list, postincrement or control addressing mode.
- 11 — Dynamic register list, postincrement or control addressing mode.

Register List field:

Static list — contains the register select mask; if a register is to be moved, the corresponding bit in the mask is set as shown below, otherwise it is clear.

Dynamic list — contains the integer data register number, rrr, as shown below:

Register List Format									
Static, $-(A_n)$	—	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0
Static, $(A_n)+$ or									
Control	—	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7
Dynamic	—	0	r	r	r	0	0	0	0

The format of the dynamic list mask is the same as for the static list and is contained in the least significant eight bits of the specified main processor data register.

Programming Note: This instruction provides a very useful feature, dynamic register list specification, that can significantly enhance system performance. If the calling conventions used for procedure calls utilize the dynamic register list feature, the number of floating-point data registers saved and restored can be reduced.

In order to utilize the dynamic register specification feature of the FMOVEM instruction, both the calling and the called procedures must be written to communicate information about register usage. When one procedure calls another, a register mask must be passed to the called procedure to indicate which registers must not be altered upon return to the calling procedure. The called procedure then saves only those registers that are modified and are already in use. There are several techniques that can be used to utilize this mechanism, and an example follows.

In this example, a convention is defined by which each called procedure is passed a word mask in D7 that identifies all floating-point registers in use by the calling procedure. Bits 15–8 identify the registers in the order FP0–FP7, and bits 7–0 identify the registers in the order FP7–FP0 (the two masks are required due to the different transfer order used by the predecrement and postincrement addressing modes). The code used by the calling procedure consists of simply moving the mask (which is generated at compile time) for the floating-point data registers currently in use into D7:

Calling procedure . . .

MOVE.W	#ACTIVE,D7	Load the list of FP registers that are in use
BSR	PROC_2	

The entry code for all other procedures computes two masks. The first mask identifies the registers in use by the calling procedure that are used by the called procedure (and therefore saved and restored by the called procedure). The second mask identifies the registers in use by the calling procedure that are used by the called procedure (and therefore not saved on entry). The appropriate registers are then stored along with the two masks:

Called procedure . . .

MOVE.W	D7,D6	Copy the list of active registers
AND.W	#WILL_USE,D7	Generate the list of doubly-used registers
FMOVEM	D7, –(A7)	Save those registers
MOVE.W	D7, –(A7)	Save the register list
EOR.W	D7,D6	Generate the list of not saved active registers
MOVE.W	D6,P(A7)	Save it for later use

If the second procedure calls a third procedure, a register mask is passed to the third procedure that indicates which registers must not be altered by the third procedure. This mask identifies any registers in the list from the first procedure that were not saved by the second procedure, plus any registers used by the second procedure that must not be altered by the third procedure.

An example of the calculation of this mask is:

Nested calling sequence . . .

MOVE.W	UNSAVED	Load the list of active registers not saved at entry
	(A7),D7	
OR.W	#ACTIVE,D7	Combine with those active at this time
BSR	PROC_3	

Upon return from a procedure, the restoration of the necessary registers follows the same convention, and the register mask generated during the save operation on entry is used to restore the required floating-point data registers:

Return to caller . . .

ADDQ.L	#2,A7	Discard the list of registers not saved
MOVE.B	(A7)+,D7	Get the saved register list (pop word, use byte)
FMOVEM	(A7)+,D7	Restore the registers
.		
.		
.		
RTS	Return to the calling routine	

Operation: Register List ∇ Destination
Source ∇ Register List

Assembler FMOVEM.L <list>,<ea>
Syntax: FMOVEM.L <ea>,<list>

<list> A list of any combination of the three floating-point system control registers (FPCR, FPSR, and FPIAR) with individual register names separated by a slash (/).

Attributes: Size = (Long)

Description: Moves one or more 32-bit values into or out of the specified system control registers. Any combination of the three system control registers may be specified. The registers are always moved in the same order, regardless of the addressing mode used; with the FPCR moved first, followed by the FPSR, and the FPIAR moved last (if a register is not selected for the transfer, the relative order of the transfer of the other registers is the same). The first register is transferred between the FPU and the specified address, with successive registers located up through higher addresses.

When more than one register is moved, the memory or memory-alterable addressing modes are allowed as shown in the addressing mode tables. If the addressing mode is predecrement, the address register is first decremented by the total size of the register images to be moved (i.e., four times the number of registers) and then the registers are transferred starting at the resultant address. For the postincrement addressing mode, the selected registers are transferred to or from the specified address, and then the address register is incremented by the total size of the register images transferred. If a single system control register is selected, the data register direct addressing mode may be used; or, if the only register selected is the FPIAR, then the address register direct addressing mode is allowed. Note that if a single register is selected, the opcode generated is the same as for the FMOVE single system control register instruction.

Floating-Point Status Register: Is changed only if the destination list includes the FPSR; in which case all bits are modified to reflect the value of the source register image.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE			REGISTER			
1	0	dr	REGISTER LIST			0	0	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field — Determines the addressing mode for the operation

Memory-to-Register — Only control addressing modes or the postincrement addressing mode are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An**	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if a single FPcr is selected.
**Only if the FPIAR is the single register selected.

Register-to-Memory — Only control alterable addressing modes or the predecrement addressing mode are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An**	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Only if a single FPcr is selected.

**Only if the FPIAR is the single register selected.

dr field — Specifies the direction of the transfer.

0 — Move the listed registers from memory to the FPU.

1 — Move the listed registers from the FPU to memory.

Register List field: — Contains the register select mask; if a register is to be moved, the corresponding bit in the list is set, otherwise it is clear.

Bit Number	—	12	11	10
Register	—	FPCR	FPSR	FPIAR

Operation: Source × FPn ↗ FPn

Assembler FMUL.<fmt> <ea>,FPn
Syntax: FMUL.X FPm,FPn
 *FrMUL<fmt> <ea>,FPn
 *FrMUL.X FPm,FPn
 where r is rounding precision, S or D
 *Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and multiplies that number by the number in the destination floating-point data register. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	Multiply	+0.0	-0.0	+inf	-inf
	-		-0.0	+0.0	-inf	+inf
Zero	+	+0.0	-0.0	+0.0	NAN ¹	
	-	-0.0	+0.0	-0.0		
Infinity	+	+inf	-inf	NAN ¹	+inf	-inf
	-	-inf	+inf		-inf	+inf

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte: BSUN Cleared
 SNAN Refer to **1.3.4 NANs**.
 OPERR Set for 0×infinity; cleared otherwise
 OVFL Refer to appropriate user’s manual for **EXCEPTION PROCESING**.

- UNFL Refer to appropriate user’s manual for **EXCEPTION PROCESSING**.
- DZ Cleared
- INEX2 Refer to appropriate user’s manual for **EXCEPTION PROCESSING**.
- INEX1 If <fmt> is Packed, refer to appropriate user’s manual for **EXCEPTION PROCESSING**; cleared otherwise.

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user’s manual.

3

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand location. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

- 000 L Long-Word Integer
- 001 S Single-Precision Real
- 010 X Extended-Precision Real
- 011 P Packed-Decimal Real*
- 100 W Word Integer
- 101 D Double-Precision Real
- 110 B Byte Integer

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FFn.

Opmode field — Specifies the instruction and rounding precision.

0100011	FMUL	Rounding precision specified by the FPCR.
1100011	FSMUL	Single-precision rounding specified.
1100111	FDMUL	Double-precision rounding specified.

FNEG

Floating-Point Negate
(MC6888x/MC68040)

FNEG

Operation: $-(\text{Source}) \nabla \text{FPn}$

Assembler FNEG.<fmt> <ea>,FPn
Syntax: FNEG.X FPm,FPn
 FNEG.X FPn
 *FrNEG.<fmt> <ea>,FPn
 *FrNEG.X FPm,FPn
 *FrNEG.X FPn
 where r is rounding precision, S or D
 *Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and inverts the sign of the mantissa. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	Negate		– 0.0	+ 0.0	– inf	+ inf

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte: BSUN Cleared
 SNAN Refer to **1.3.4 NANs**.
 OPERR Cleared
 OVFL Cleared
 UNFL If source is an extended precision denormalized number, refer to appropriate user’s manual for **EXCEPTION PROCESSING**; cleared otherwise.
 DZ Cleared
 INEX2 Cleared

INEX1 If <fmt> is Packed, refer to appropriate user's manual for **EXCEPTION PROCESSING**; cleared otherwise.

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#{data}	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M=0, specifies the source floating-point data register, FPM.

If R/M=1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding will cause an unimplemented data type exception to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M=0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Opmode field — Specifies the instruction and rounding precision.

0011010	FNEG	Rounding precision specified by the FPCR.
1011010	FSNEG	Single-precision rounding specified.
1011110	FDNEG	Double-precision rounding specified.

FNOP

No Operation
(MC6888x/MC68040)

FNOP

Operation: None

Assembler

Syntax: FNOP

Attributes: Unsize

Description: This instruction does not perform any explicit operation. However, it is useful to force synchronization of the FPU with an integer unit, or to force processing of pending exceptions. For most floating-point instructions, the integer unit is allowed to continue with the execution of the next instruction once the FPU has any operands needed for an operation, thus supporting concurrent execution of floating-point and integer instructions. However, the FNOP instruction synchronizes the FPU and the IU by causing the IU to wait until all previous floating-point instructions have completed. Execution of FNOP also forces any exceptions pending from the execution of a previous floating-point instruction to be processed as a preinstruction exception.

The MC68882 may not wait to begin execution of another floating-point instruction until it has completed execution of the current instruction. However, the FNOP instruction synchronizes the coprocessor and MPU by causing the MPU to wait until the current instruction (or both instructions) have completed.

The FNOP instruction also forces the processing of exceptions pending from the execution of previous instructions. This is also inherent in the way that the FPCP utilizes the M68000 Family coprocessor interface. Once the FPCP has received the input operand for an arithmetic instruction, it always releases the main processor to execute the next instruction (regardless of whether or not concurrent execution is prevented for the instruction due to tracing) without reporting the exception during the execution of that instruction. Then, when the main processor attempts to initiate the execution of the next FPCP instruction, a pre-instruction exception may be reported to initiate exception processing for an exception that occurred during a previous instruction. By using the FNOP instruction, the user can force any pending exceptions to be processed without performing any other operations.

Status Register: Not Affected

FNOP

No Operation
(MC6888x/MC68040)

FNOP

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

NOTE

FNOP uses the same opcode as the FBcc.W <label> instruction, with cc = F (nontrapping false) and <label> = * + 2 (which results in a displacement of 0).

FREM

IEEE Remainder
(MC6888x/040SW)

FREM

Operation: IEEE Remainder of (FPn (÷) Source) ÷ FPn

Assembler FREM.<fmt> <ea>,FPn

Syntax: FREM.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the modulo remainder of the number in the destination floating-point data register, using the source operand as the modulus. Stores the result in the destination floating-point data register, and stores the sign and seven least significant bits of the quotient in the FPSR quotient byte (the quotient is the result of FPn (÷) Source). The IEEE remainder function is defined as:

$$FPn - (Source \times N)$$

where:

$N = \text{INT} (FPn (\div) \text{Source})$ in the round-to-nearest mode

The FREM function is not defined for a source operand equal to zero or for a destination operand equal to infinity. Note that this function is not the same as the FMOD instruction, which uses the round-to-zero mode and thus returns a remainder that is different from the remainder required by the *IEEE Specification for Binary Floating-Point Arithmetic*.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	-	IEEE Remainder		FPn ²	
Zero	+	-	+0.0 -0.0		+0.0 -0.0	
	+	-	NaN ¹		NaN ¹	
Infinity	+	-	NaN ¹		NaN ¹	
	+	-	NaN ¹		NaN ¹	

NOTES:

1. Sets the OPERR bit in the FPSR exception byte.
2. Returns the value of FPn before the operation. However, the result is processed by the normal instruction termination procedure to round it as required. Thus, an underflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FPn value was loaded.
3. If either operand is a NaN, refer to **1.3.4 NaNs** for more information.

Status Register:

- Condition Codes:
- Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**
- Quotient Byte:
- Loaded with the sign and least significant seven bits of the quotient (FPn (÷) Source). The sign of the quotient is the exclusive OR of the sign bits of the source and destination operands.
- Exception Byte:
- BSUN

Cleared

SNAN

Refer to **1.3.4 NaNs**.

OPERR

Set if the source is zero, or the destination is infinity; cleared otherwise

OVFL

Cleared

UNFL

Refer to appropriate user's manual for **Underflow**.

DZ

Cleared

INEX2

Cleared

INEX1

If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
							MODE			REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	0	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if {fmt} is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN.

FSCALE

Scale Exponent
(MC6888x/040SW)

FSCALE

Operation: $FP_n \times INT(2^{Source}) \rightarrow FP_n$

Assembler FSCALE.<fmt> <ea>,FP_n

Syntax: FSCALE.X FP_m,FP_n

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to an integer (if necessary) and adds that integer to the destination exponent. Stores the result in the destination floating-point data register. This function has the effect of multiplying the destination by 2^{Source} , but is much faster than a multiply operation when the source is an integer value.

The FPCP assumes that the scale factor is an integer value before the operation is executed. If not, the value is chopped (i.e., rounded using the round-to-zero mode) to an integer before it is added to the exponent. When the absolute value of the source operand is $(\geq) 2^{14}$, an overflow or underflow always results.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	-	Scale Exponent		FP _n ¹	
Zero	+	-	+ 0.0 - 0.0		NAN ²	
Infinity	+	-	+ inf - inf		NAN ²	

NOTES:

1. Returns the value FP_n before the operation. However, the result is processed by the normal instruction termination procedure to round it as required. Thus, an underflow and/or inexact result may occur if the rounding precision has been changed to a smaller size since the FP_n value was loaded.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NANs .
OPERR	Set if the source operand is (+ or -)infinity; cleared otherwise
OVFL	Refer to appropriate user's manual for Overflow .
UNFL	Refer to appropriate user's manual for Underflow .
DZ	Cleared
INEX2	Cleared
INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE				REGISTER	
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	1	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

- 000 L Long-Word Integer
- 001 S Single-Precision Real
- 010 X Extended-Precision Real
- 011 P Packed-Decimal Real
- 100 W Word Integer
- 101 D Double-Precision Real
- 110 B Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN.

Operation: If (condition true)
 then 1s → Destination
 else 0s → Destination

Assembler

Syntax: FScc.<size> <ea>

Attributes: Size = (Byte)

Description: If the specified floating-point condition is true, sets the byte integer operand at the destination to TRUE (all ones), otherwise sets the byte to FALSE (all zeros). The conditional specifier cc may select any one of the 32 floating-point conditional tests as described in **1.3.2 Conditional Test Definitions**.

3

Floating-Point Status Register:

Condition Codes: Not affected

Quotient Byte: Not affected

Exception Byte: BSUN Set if the NAN condition code is set and the condition selected is an IEEE nonaware test

 SNAN Not Affected

 OPERR Not Affected

 OVFL Not Affected

 UNFL Not Affected

 DZ Not Affected

 INEX2 Not Affected

 INEX1 Not Affected

Accrued Exception Byte: The IOP bit is set if the BSUN bit is set in the exception byte. No other bit is affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICATE					

Instruction Fields:

Effective Address field — Specifies the addressing mode for the byte integer operand. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Conditional Predicate field — Specifies one of 32 conditional tests as defined in 1.3.2 Conditional Test Definitions.

NOTE

When a BSUN exception occurs, a pre-instruction exception is taken. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FSc), then it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception occurs again immediately upon return to the routine that caused the exception.

FSGLDIV

Single Precision Divide
(MC6888x/040SW)

FSGLDIV

Operation: $FPn (\div) \text{Source} \blacktriangleright FPn$

Assembler FSGLDIV.<fmt> <ea>,FPn

Syntax: FSGLDIV.X FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and divides that number into the number in the destination floating-point data register. Stores the result in the destination floating-point data register, rounded to single precision (regardless of the current rounding precision). This function is undefined for $0(\div)0$ and infinity(\div)infinity.

Both the source and destination operands are assumed to be representable in the single precision format. If either operand requires more than 24 bits of mantissa to be accurately represented, the accuracy of the result is not guaranteed. Furthermore, the result exponent may exceed the range of single precision, regardless of the rounding precision selected in the FPCR mode control byte. Refer to **1.3.5.2 UNDERFLOW, ROUND, OVERFLOW** for more information.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	-	Divide (Single Precision)	$+inf^1$ $-inf^1$	$-inf^1$ $+inf^1$	$+0.0$ -0.0
Zero	+	-	$+0.0$ -0.0	-0.0 $+0.0$	NAN ²	$+0.0$ -0.0
Infinity	+	-	$+inf$ $-inf$	$-inf$ $+inf$	$+inf$ $-inf$	$-inf$ $+inf$

NOTES:

1. Sets the DZ bit in the FPSR exception byte.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NaNs .
	OPERR	Set for $0(\div)0$ or $\text{infinity}(\div)\text{infinity}$
	OVFL	Refer to appropriate user's manual for Overflow .
	UNFL	Refer to appropriate user's manual for Underflow .
	DZ	Set if the source is zero and the destination is in range; cleared otherwise
	INEX2	Refer to appropriate user's manual for Inexact Result .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	0	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
–(An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPN.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN.

FSGLMUL

Single Precision Multiply
(MC6888x/040SW)

FSGLMUL

Operation: Source × FPn ↗ FPn

Assembler FSGLMUL.<fmt>
Syntax: FSGLMUL.X
 <ea>,FPn
 FPm,FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and multiplies that number by the number in the destination floating-point data register. Stores the result in the destination floating-point data register, rounded to single precision (regardless of the current rounding precision).

3

Both the source and destination operands are assumed to be representable in the single precision format. If either operand requires more than 24 bits of mantissa to be accurately represented, the accuracy of the result is not guaranteed. Furthermore, the result exponent may exceed the range of single precision, regardless of the rounding precision selected in the FPCR mode control byte. Refer to **1.3.5.2 UNDERFLOW, ROUND, OVERFLOW** for more information.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	−	+	−	+	−
In Range	+	Multiply (Single Precision)	+0.0	−0.0	+inf	−inf
	−		−0.0	+0.0	−inf	+inf
Zero	+	+0.0	−0.0	+0.0	NAN ¹	
	−	−0.0	+0.0	−0.0		
Infinity	+	+inf	−inf	NAN ¹	+inf	−inf
	−	−inf	+inf		−inf	+inf

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Set if one operand is zero and the other is infinity; cleared otherwise
	OVFL	Refer to appropriate user's manual for Overflow .
	UNFL	Refer to appropriate user's manual for Underflow .
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for Inexact Result .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE	REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	1	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

FSGLMUL

Single Precision Multiply

(MC6888x/040SW)

FSGLMUL

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPN.

If R/M = 1, specifies the source data format:

000

L

Long-Word Integer

001

S

Single-Precision Real

010

X

Extended-Precision Real

011

P

Packed-Decimal Real

100

W

Word Integer

101

D

Double-Precision Real

110

B

Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN.

Operation: Sine of Source ♦ FPN

Assembler FSIN.<fmt> <ea>,FPn
Syntax: FSIN.X FPM,FPn
FSIN.X FPN

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the sine of that number. Stores the result in the destination floating-point data register. This function is not defined for source operands of (\pm)infinity. If the source operand is not in the range of $[-2\pi \dots +2\pi]$, the argument is reduced to within that range before the sine is calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) lose all accuracy. The result is in the range of $[-1 \dots +1]$.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Sine		+0.0	-0.0	NaN ¹	

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NANs .
OPERR	Set if the source is (+ or -)infinity; cleared otherwise.
OVFL	Cleared
UNFL	Refer to appropriate user's manual for Underflow .
DZ	Cleared

- INEX2
- Refer to appropriate user’s manual for **Inexact Result**.
- INEX1
- If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE				REGISTER		
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	1	1	1	1	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, then the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FSINCOS

Simultaneous Sine and Cosine
(MC6888x/040SW)

FSINCOS

Operation:

Sine of Source \blacklozenge FPs
Cosine of Source \blacklozenge FPc

Assembler

FSINCOS.<fmt>

<ea>,FPc:FPs

Syntax:

FSINCOS.X

FPm,FPc:FPs

Attributes:

Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description:

Converts the source operand to extended precision (if necessary) and calculates both the sine and the cosine of that number. Calculates both functions simultaneously; thus, this instruction is significantly faster than performing separate FSIN and FCOS instructions. Loads the sine result into the destination floating-point data register, FPs, and the cosine result into the destination floating-point data register FPc. Sets the condition code bits according to the sine result. If FPs and FPc are specified to be the same register, the cosine result is first loaded into the register and then is overwritten with the sine result. This function is not defined for source operands of (\pm)infinity.

If the source operand is not in the range of $(-2\pi \dots +2\pi]$, the argument is reduced to within that range before the sine and cosine are calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) lose all accuracy. The results are in the range of $[-1 \dots +1]$.

Operation Table:

<div> <div>Source</div> <div>Destination</div> </div>	In Range		Zero		Infinity	
	+	-	+	-	+	-
FPs	Sine		+ 0.0	- 0.0	NaN ¹	
FPc	Cosine		+ 1.0		NaN ¹	

- NOTES:
1. Sets the OPERR bit in the FPSR exception byte.

2. If the source operand is a NaN, refer to **1.3.4 NaNs** for more information.

Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES (for the sine result)	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NaNs .
	OPERR	Set if the source is (\pm)infinity; cleared otherwise
	OVFL	Cleared
	UNFL	Set if a sine underflow occurs, in which case the cosine result is 1. Cosine cannot underflow. Refer to appropriate user's manual for Underflow .
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for Inexact Result .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER, FPs			0	1	1	0	DESTINATION REGISTER, FPs		

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.
If R/M = 0, this field is unused, and should be all zeros.
If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPM.
If R/M = 1, specifies the source data format:

- 000 L Long-Word Integer
- 001 S Single-Precision Real
- 010 X Extended-Precision Real
- 011 P Packed-Decimal Real
- 100 W Word Integer
- 101 D Double-Precision Real
- 110 B Byte Integer

Destination Register, FPC field — Specifies the destination floating-point data register, FPC. The cosine result is stored in this register.

Destination Register, FPs field — Specifies the destination floating-point data register, FPs. The sine result is stored in this register. If FPc and FPs specify the same floating-point data register, the sine result is stored in the register, and the cosine result is discarded.

If R/M=0 and the source register field is equal to either of the destination register fields, the input operand is taken from the specified floating-point data register, and the appropriate result is written into the same register.

Operation: Hyperbolic Sine of Source ∇ FPn

Assembler FSINH.<fmt> <ea>,FPn
Syntax: FSINH.X FPm,FPn
FSINH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the hyperbolic sine of that number. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Hyperbolic Sine		+0.0	-0.0	+inf	-inf

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

- BSUN Cleared
- SNAN Refer to **1.3.4 NANs**.
- OPERR Cleared
- OVFL Refer to appropriate user's manual for **Overflow**.
- UNFL Refer to appropriate user's manual for **Underflow**.
- DZ Cleared
- INEX2 Refer to appropriate user's manual for **Inexact Result**.
- INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	1	0

3

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(d8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([w,d,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(d8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <mt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPm.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPn. If R/M = 0 and the source and destination fields are equal, the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: Square Root of Source ♦ FPn

Assembler FSQRT.<fmt> <ea>,FPn
Syntax: FSQRT.X FPm,FPn
FSQRT.X FPn
*FrSQRT.<fmt> <ea>,FPn
*FrSQRT FPm,FPn
*FrSQRT FPn
where r is rounding precision, S or D
*Supported by MC68040 only)

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the square root of that number. Stores the result in the destination floating-point data register. This function is not defined for negative operands.

FSQRT will round the result to the precision selected in the floating-point control register. FSFSQRT and FDFSQRT will round the result to single or double precision respectively, regardless of the rounding precision selected in the FPCR.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	–	+	–	+	–
Result	\sqrt{x}	NAN ¹	+0.0	–0.0	+inf	NAN ¹

- NOTES:
- 1. Sets the OPERR bit in the FPSR exception byte.
 - 2. If the source operand is a NAN, refer to 1.3.4 NANs for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES

Quotient Byte: Not affected

Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NaNs .
	OPERR	Set if the source operand is not zero and is negative; cleared otherwise
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

FSQRT

Floating-Point Square Root
(MC6888x/MC68040)

FSQRT

Destination Register field — Specifies the destination floating-point data register, FFn. If R/M = 0 and the source and destination fields are equal, the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Opmode field — Specifies the instruction and rounding precision.

0000100	FSQRT	Rounding precision specified by the FPCR.
1000001	FSSQRT	Single-precision rounding specified.
1000101	FDSQRT	Double-precision rounding specified.

Operation: $FPn - \text{Source} \rightarrow FPn$

Assembler Syntax: FSUB.<fmt> <ea>,FPn
 FSUB.X FPm,FPn
 *FrSUB.<fmt> <ea>,FPn
 *FrSUB.X FPm,FPn
 where r is rounding precision, S or D
 *Supported by MC68040 only

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and subtracts that number from the number in the destination floating-point data register. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
In Range	+	-	Subtract		- inf	+ inf
Zero	+	-	Subtract		- inf	+ inf
			+ 0.0 ¹	- 0.0		
			- 0.0	+ 0.0 ¹		
Infinity	+	-	+ inf		NAN ²	- inf
			- inf		- inf	NAN ²

NOTES:

1. Returns +0.0 in rounding modes RN, RZ, and RP; returns -0.0 in RM.
2. Sets the OPERR bit in the FPSR exception byte.
3. If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Floating-Point Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NaNs .
	OPERR	Set if both the source and destination are like-signed infinities; cleared otherwise
	OVFL	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	UNFL	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	DZ	Cleared
	INEX2	Refer to appropriate user's manual for EXCEPTION PROCESSING .
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real*
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

Destination Register field — Specifies the destination floating-point data register, FPN.

Opmode field — Specifies the instruction and rounding precision.

0101000	FSUB	Rounding precision specified by the FPCR.
1101000	FSSUB	Single-precision rounding specified.
1101100	FDSUB	Double-precision rounding specified.

Operation: Tangent of Source \rightarrow FPn

Assembler FTAN.<fmt> <ea>,FPn
Syntax: FTAN.X FPm,FPn
 FTAN.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the tangent of that number. Stores the result in the destination floating-point data register. This function is not defined for source operands of $(\pm)\text{infinity}$. If the source operand is not in the range of $[-\pi/2 \dots +\pi/2]$, the argument is reduced to within that range before the tangent is calculated. However, large arguments may lose accuracy during reduction, and very large arguments (greater than approximately 10^{20}) lose all accuracy.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	Tangent		+0.0	-0.0	NaN ¹	

NOTES:

1. Sets the OPERR bit in the FPSR exception byte.
2. If the source operand is a NaN, refer to **1.3.4 NaNs** for more information.

Status Register:

Condition Codes: Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NaNs .
OPERR	Set if the source is $(\pm)\text{infinity}$; cleared otherwise
OVFL	Refer to appropriate user's manual for Overflow .
UNFL	Refer to appropriate user's manual for Underflow .
DZ	Cleared

- INEX2 Refer to appropriate user's manual for **Inexact Result**.
- INEX1 If <fmt> is Packed, refer to appropriate user's manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	1	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.
 If R/M = 0, this field is unused, and should be all zeros.
 If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
((bd,An,Xn),od)	110	reg. number:An
((bd,An),Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
. #<data>	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
((bd,PC,Xn),od)	111	011
((bd,PC),Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FTANH

Hyperbolic Tangent
(MC6888x/040SW)

FTANH

Operation: Hyperbolic Tangent of Source ♦ FPn

Assembler FTANH.<fmt> <ea>,FPn
Syntax: FTANH.X FPm,FPn
FTANH.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates the hyperbolic tangent of that number. Stores the result in the destination floating-point data register.

3

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	−	+	−	+	−
Result	Hyperbolic Tangent		+ 0.0	− 0.0	+ 1.0	− 1.0

NOTE: If the source operand is a NAN, refer to **1.3.4 NANs** for more information.

Status Register:

Condition Codes:Affected as described in **1.3.5.1 SETTING FLOATING-POINT CONDITION CODES**

Quotient Byte:Not affected

Exception Byte:

BSUNCleared

SNANRefer to **1.3.4 NANs**.

OPERRCleared

OVFLCleared

UNFLRefer to appropriate user’s manual for **Under-flow**.

DZCleared

INEX2Refer to appropriate user’s manual for **Inexact Result**.

INEX1If <fmt> is Packed, refer to appropriate user’s manual for **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	0	0	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

FTENTOX

10^X
(MC6888x/MC68040SW)

FTENTOX

Operation: 10Source → FPn

Assembler FTENTOX.<fmt> <ea>,FPn
Syntax: FTENTOX.X FPm,FPn
FTENTOX.X FPn

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates 10 to the power of that number. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	10 ^X		+ 1.0		+ inf	+ 0.0

NOTE: If the source operand is a NAN, refer to 1.3.4 NANs for more information.

Status Register:

Condition Codes: Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES

Quotient Byte: Not affected

Exception Byte:

- BSUN Cleared
- SNAN Refer to 1.3.4 NANs.
- OPERR Cleared
- OVFL Refer to appropriate user’s manual for **Overflow**.
- UNFL Refer to appropriate user’s manual for **Underflow**.
- DZ Cleared
- INEX2 Refer to appropriate user’s manual for **Inexact Result**.
- INEX1 If <fmt> is Packed, refer to appropriate user’s manual **Inexact Result on Decimal Input**; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	1	0

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
[(bd,An,Xn),od)	110	reg. number:An
[(bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
[(bd,PC,Xn),od)	111	011
[(bd,PC],Xn,od)	111	011

*Only if {fmt} is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

Operation: If condition true, then TRAP

Assembler FTRAPcc

Syntax: FTRAPcc.W #<data>
FTRAPcc.L #<data>

Attributes: Size = (Word, Long)

Description: If the selected condition is true, the processor initiates exception processing. A vector number is generated to reference the TRAPcc exception vector. The stacked program counter points to the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction in sequence. The immediate data operand is placed in the word(s) following the conditional predicate word and is available for user definition for use within the trap handler.

The conditional specifier cc selects one of the 32 conditional tests defined in **4.4.2 Conditional Test Definitions**.

Floating-Point Status Register:

Condition Codes:	Not affected	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Set if the NAN condition code is set and the condition selected is an IEEE nonaware test
	SNAN	Not Affected
	OPERR	Not Affected
	OVFL	Not Affected
	UNFL	Not Affected
	DZ	Not Affected
	INEX2	Not Affected
	INEX1	Not Affected

Accrued Exception Byte: The IOP bit is set if the BSUN bit is set in the exception byte. No other bit is affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	1	1	1	1	MODE		
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICATE					
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)															
LEAST SIGNIFICANT WORD OR 32-BIT OPERAND (IF NEEDED)															

Instruction Fields:

Mode field — Specifies the form of the instruction.

010 — The instruction is followed by a word operand.

011 — The instruction is followed by a long-word operand.

100 — The instruction has no operand.

Conditional Predicate field — Specifies one of 32 conditional tests as described in **1.3.2 Conditional Test Definitions**.

Operand field — Contains an optional word or long-word operand that is user defined.

NOTE

When a BSUN exception occurs, a pre-instruction exception is taken by the main processor. If the exception handler returns without modifying the image of the PC on the stack frame (to point to the instruction following the FTRAPcc), it must clear the cause of the exception (by clearing the NAN bit or disabling the BSUN trap) or the exception occurs again immediately upon return to the routine that caused the exception.

Operation: Condition Codes for Operand ♦ FPCC

Assembler FTST.<fmt> <ea>
Syntax: FTST.X FPM

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and sets the condition code bits according to the data type of the result.

Operation Table: The contents of this table differ from the other operation tables. A letter in an entry of this table indicates that the designated condition code bit is always set by the FTST operation. All unspecified condition code bits are cleared during the operation.

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	none	N	Z	NZ	I	NI

- NOTES:
- 1. If the source operand is a NAN, set the NAN condition code bit.
 - 2. If the source operand is a SNAN, set the SNAN bit in the FPSR exception byte.

Floating-Point Status Register:

Condition Codes:	Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES	
Quotient Byte:	Not affected	
Exception Byte:	BSUN	Cleared
	SNAN	Refer to 1.3.4 NANs .
	OPERR	Cleared
	OVFL	Cleared
	UNFL	Cleared
	DZ	Cleared
	INEX2	Cleared
	INEX1	If <fmt> is Packed, refer to appropriate user's manual for EXCEPTION PROCESSING ; cleared otherwise.

Accrued Exception Byte: Affected as described in **EXCEPTION PROCESSING**, refer to appropriate user’s manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE			REGISTER			
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	1	1	0	1	0

3

Instruction Fields:

Effective Address field — Determines the addressing mode for external operands.
If R/M=0, this field is unused, and should be all zeros.
If R/M=1, specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if (fmt) is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.
0 — The operation is register to register.
1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.
If R/M = 0, specifies the source floating-point data register, FPM.
If R/M = 1, specifies the source data format:

- 000 L Long-Word Integer
- 001 S Single-Precision Real
- 010 X Extended-Precision Real
- 011 P Packed-Decimal Real*
- 100 W Word Integer
- 101 D Double-Precision Real
- 110 B Byte Integer

*This encoding will cause an unimplemented data type exception in the MC68040 to allow emulation in software.

Destination Register field — Since the FPU uses a common command word format for all of the arithmetic instructions (including FTST), this field is treated in the same manner for FTST as for the other arithmetic instructions, even though the destination register is not modified. This field should be set to zero in order to maintain compatibility with future devices, although the FPU does not signal an illegal instruction trap if it is not zero.

Operation: 2Source ∇ FPN

Assembler FTWOTOX.<fmt> <ea>,FPn
Syntax: FTWOTOX.X FPM,FPn
FTWOTOX.X FPN

Attributes: Format = (Byte, Word, Long, Single, Double, Extended, Packed)

Description: Converts the source operand to extended precision (if necessary) and calculates two to the power of that number. Stores the result in the destination floating-point data register.

Operation Table:

Destination \ Source	In Range		Zero		Infinity	
	+	-	+	-	+	-
Result	2 ^x		+ 1.0		+ inf	+ 0.0

NOTE: If the source operand is a NAN, refer to 1.3.4 NANs for more information.

Status Register:

Condition Codes: Affected as described in 1.3.5.1 SETTING FLOATING-POINT CONDITION CODES

Quotient Byte: Not affected

Exception Byte:

BSUN	Cleared
SNAN	Refer to 1.3.4 NANs.
OPERR	Cleared
OVFL	Refer to appropriate user's manual for Overflow .
UNFL	Refer to appropriate user's manual for Underflow .
DZ	Cleared
INEX2	Refer to appropriate user's manual for Inexact Result .
INEX1	If <fmt> is Packed, refer to appropriate user's manual for Inexact Result on Decimal Input ; cleared otherwise.

Accrued Exception Byte: Affected as described in **IEEE Exception and Trap Compatibility**, refer to appropriate user's manual.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	0	1

Instruction Fields:

Coprocessor ID field — Specifies which coprocessor in the system is to execute this instruction. Motorola assemblers default to ID = 1 for the FPCP.

Effective Address field — Determines the addressing mode for external operands.

If R/M = 0, this field is unused, and should be all zeros.

If R/M = 1, this field is encoded with an M68000 addressing mode as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Only if <fmt> is Byte, Word, Long, or Single.

R/M field — Specifies the source operand address mode.

0 — The operation is register to register.

1 — The operation is <ea> to register.

Source Specifier field — Specifies the source register or data format.

If R/M = 0, specifies the source floating-point data register, FPM.

If R/M = 1, specifies the source data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

Destination Register field — Specifies the destination floating-point data register, FPN. If R/M = 0 and the source and destination fields are equal, the input operand is taken from the specified floating-point data register, and the result is written into the same register. If the single register syntax is used, Motorola assemblers set the source and destination fields to the same value.

SECTION 4

SUPERVISOR (PRIVILEGED) INSTRUCTIONS

This section contains detailed information about the supervisor privileged instructions for the M68000 Family. Each instruction is described in detail and the instruction descriptions are arranged in alphabetical order by instruction mnemonic.

Any differences within the M68000 Family of instructions is identified in the instruction. If an instruction only applies to a certain processor or processors, the processor(s) that the instruction pertains to is identified under the title of the instruction. For example:

RESET
(MC68030/MC68040)

If the instruction applies to all the M68000 Family but a processor or processors may use a different instruction field, instruction format, etc., the differences will be identified within the paragraph. For example:

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)	110	reg. number:An
------------	-----	----------------

The following instructions are listed separately for each processor due to the many differences involved within the instruction:

- PFLUSH Flush ATC Entries
- PMOVE Move PMMU (MMU) Register
- PTEST Test a Logical Address

Appendix A provides a listing of all processors and the instructions that apply to them for quick reference.

ANDI to SR

AND Immediate to the Status Register
(M68000 Family)

ANDI to SR

Operation: If supervisor state
then Source \wedge SR \rightarrow SR
else TRAP

Assembler

Syntax: ANDI #<data>,SR

Attributes: Size = (Word)

Description: Performs an AND operation of the immediate operand with the contents of the status register and stores the result in the status register. All implemented bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Cleared if bit 4 of immediate operand is zero. Unchanged otherwise.
- N Cleared if bit 3 of immediate operand is zero. Unchanged otherwise.
- Z Cleared if bit 2 of immediate operand is zero. Unchanged otherwise.
- V Cleared if bit 1 of immediate operand is zero. Unchanged otherwise.
- C Cleared if bit 0 of immediate operand is zero. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
WORD DATA (16 BITS)															

Operation: If supervisor state
then invalidate selected cache lines
else TRAP

Assembler CINVL <cache>, (An)
Syntax: CINV <cache>, (An)
CINVA <cache>

<cache> specifies either the instruction (IC), data (DC),
or both caches.

Attributes: Unsized

Description: Invalidates selected cache lines. The data cache (DC), instruction cache (IC), both caches (BC), or neither cache (NC) can be specified. Any dirty data in data cache lines that are invalidated is lost; the CPUSH instruction must be used when dirty data may be contained in the data cache.

Specific cache lines can be selected in three ways:

- 1) CINVL invalidates the cache line (if any) matching the physical address in the specified address register.
- 2) CINV <cache>, (An) invalidates the cache lines (if any) matching the physical memory page in the specified address register. For example, if 4K page sizes are selected and An contains \$12345000, all cache lines matching page \$12345000 are invalidated.
- 3) CINVA invalidates all cache entries.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	CACHE		0	SCOPE		REGISTER		

Instruction Fields:

Cache field — Specifies the Cache:

- 00 — No Operation
- 01 — Data Cache
- 10 — Instruction Cache
- 11 — Data and Instruction Caches

Scope field — Specifies the Scope of the Operation:

- 00 — Illegal (causes illegal instruction trap)
- 01 — Line
- 10 — Page
- 11 — All

Register field — Specifies the address register for line and page operations.
For line operations, the low order bits 3:0 of the address are don't cares.
Bits 11:0 or 12:0 of the address are don't care for 4K or 8K page operations, respectively.

cpRESTORE

Coprocessor Restore
Functions
(MC68020/MC68030)

cpRESTORE

Operation: If supervisor state
then Restore Internal State of Coprocessor
else TRAP

Assembler

Syntax: cpRESTORE (ea)

Attributes: Unsized

Description: Restores the internal state of a coprocessor usually after it has been saved by a preceding cpSAVE instruction.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			1	0	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Instruction Field:

Cp-Id field — Identifies the coprocessor that is to be restored. Cp-Id of 000 results in an F-line exception for the MC68030.

Effective Address field — Specifies the location where the internal state of the coprocessor is located. Only postincrement or control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
(An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

NOTE

If the format word returned by the coprocessor indicates “come again”, pending interrupts are not serviced.

Operation: If supervisor state
then Save Internal State of Coprocessor
else TRAP

Assembler

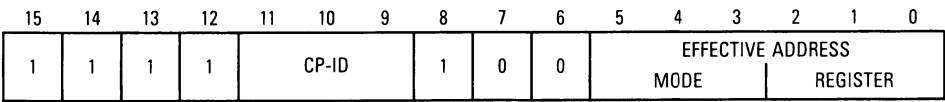
Syntax: cpSAVE <ea>

Attributes: Unsized

Description: Saves the internal state of a coprocessor in a format that can be restored by a cpRESTORE instruction.

Condition Codes:
Not affected.

Instruction Format:



Instruction Fields:

Cp-ld field — Identifies the coprocessor for this operation. Cp-ld of 000 results in an F-line exception for the MC68030.

Effective Address field — Specifies the location where the internal state of the coprocessor is to be saved. Only predecrement or control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An*	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Operation: If supervisor state
 then
 if data cache then push selected dirty data cache lines
 invalidate selected cache lines
 else TRAP

Assembler CPUSHL <cache>, (An)
Syntax: CPUSHP <cache>, (An)
 CPUSHA <cache>

<cache> specifies either the instruction (IC), data (DC), or both caches.

Attributes: Unsize

Description: Pushes and then invalidates selected cache lines. The data cache (DC), instruction cache (IC), both caches (BC), or neither cache (NC) can be specified. When the data cache is specified, the selected data cache lines are first pushed to memory (if they contain dirty data) and then invalidated. Selected instruction cache lines are invalidated.

4

Specific cache lines can be selected in three ways:

- 1) CPUSHL pushes and invalidates the cache line (if any) matching the physical address in the specified address register.
- 2) CPUSHP pushes and invalidates the cache lines (if any) matching the physical memory page in the specified address register. For example, if 4K page sizes are selected and An contains \$12345000, all cache lines matching page \$12345000 are selected.
- 3) CPUSHA pushes and invalidates all cache entries.

Condition Codes:
 Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	CACHE	1	SCOPE	REGISTER				

Instruction Fields:

Cache field — Specifies the Cache:

- 00 — No Operation
- 01 — Data Cache
- 10 — Instruction Cache
- 11 — Data and Instruction Caches

Scope field — Specifies the Scope of the Operation:

- 00 — Illegal (causes illegal instruction trap)
- 01 — Line
- 10 — Page
- 11 — All

Register field — Specifies the address register for line and page operations.
For line operations, the low order bits 3:0 of the address are don't care.
Bits 11:0 or 12:0 of the address are don't care for 4K or 8K page operations, respectively.

EORI to SR

Exclusive-OR Immediate to the Status Register
(M68000 Family)

EORI to SR

Operation: If supervisor state
then Source \oplus SR \rightarrow SR
else TRAP

Assembler

Syntax: EORI #<data>,SR

Attributes: Size = (Word)

Description: Performs an exclusive-OR operation on the contents of the status register using the immediate operand and stores the result in the status register. All implemented bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Changed if bit 4 of immediate operand is one. Unchanged otherwise.
- N Changed if bit 3 of immediate operand is one. Unchanged otherwise.
- Z Changed if bit 2 of immediate operand is one. Unchanged otherwise.
- V Changed if bit 1 of immediate operand is one. Unchanged otherwise.
- C Changed if bit 0 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
WORD DATA (16 BITS)															

FRESTORE

Restore Internal
Floating-Point State
(MC68040/MC68881/MC68882)

FRESTORE

Operation: If in supervisor state
then FPU State Frame → Internal State
else TRAP

Assembler

Syntax: FRESTORE <ea>

Attributes: Unsized

Description: Aborts the execution of any floating-point operation in progress, and loads a new FPU internal state from the state frame located at the effective address. The first word at the specified address is the format word of the state frame, which specifies the size of the frame and the revision number of the FPU that created it. If the format word is invalid (either because the size of the frame is not recognized, or the revision number does not match the revision of the FPU), the FRESTORE is aborted and a format exception is generated. If the format word is valid, the appropriate state frame is loaded, starting at the specified location and proceeding through higher addresses.

4

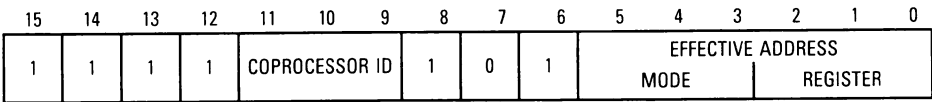
The FRESTORE instruction does not normally affect the programmer's model registers of the FPCP (except for the NULL state size, as described below), but is used only to restore the user invisible portion of the machine. The FRESTORE instruction is used with the FMOVEM instruction to perform a full context restoration of the FPU, including the floating-point data registers and system control registers. In order to accomplish a complete restoration, the FMOVEM instructions are first executed to load the programmer's model, followed by the FRESTORE instruction to load the internal state and continue any previously suspended operation.

The current implementation supports three state frames. These state frames are defined as follows:

- NULL:** This state frame is four bytes long, with a format word of \$0000. An FRESTORE operation with this size state frame is equivalent to a hardware reset of the FPU. The programmer's model is set to the reset state, with nonsignaling NaNs in the floating-point data registers and zeros in the FPCR, FPSR, and FPIAR. (Thus, it is unnecessary to load the programmer's model before this operation.)
- IDLE:** This state frame is four bytes long in the MC68040, 28 (\$1C) bytes long in the MC68881 and 60 (\$3C) bytes long in the MC68882. An FRESTORE operation with this state frame causes the FPU to be restored to the idle state, waiting for the initiation of the next instruction, with no exceptions pending. The programmer's model is not affected by loading this type of state frame.
- BUSY:** This state frame is 96 (\$60) bytes long in the MC68040, 184 (\$B8) bytes long in the MC68881 and 216 (\$D8) bytes long in the MC68882. An FRESTORE operation with this size state frame causes the FPU to be restored to the busy state, executing the instructions that were suspended by a previous FSAVE operation. The programmer's model is not affected by loading this type of state frame (although the completion of the suspended instructions after the restore is executed may modify the programmer's model).

Floating-Point Status Register: Cleared if the state size is NULL, otherwise not affected

Instruction Format:



Instruction Fields:

Effective Address field — Determines the addressing mode for the state frame.
Only postincrement or control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

FSAVE

Save Internal Floating-Point State (MC68040/MC68881/MC68882)

FSAVE

Operation: If in supervisor state
then FPU Internal State ♦ State Frame
else TRAP

Assembler

Syntax: FSAVE <ea>

Attributes: Unsized

Description: For the **MC68040** FSAVE allows the completion of any floating-point operation in progress, and saves the internal state of the FPU in a state frame located at the effective address. After the save operation, the FPU is in the idle state, waiting for the execution of the next instruction. The first word written to the state frame is the format word, which specifies the size of the frame and the revision number of the FPU.

Any floating-point operations in progress when an FSAVE instruction is encountered are allowed to complete before the FSAVE is executed, to allow saving an IDLE state frame if possible. Execution of instructions already in the FPU pipeline continues until either completion of all instructions in the pipeline or generation of an exception by one of the instructions. An IDLE state frame will be created by the FSAVE if no exceptions occurred; otherwise a BUSY stack frame is created.

For the **MC68881/MC68882** FSAVE suspends the execution of any operation in progress, and saves the internal state in a state frame located at the effective address. After the save operation, the FPCP is in the idle state, waiting for the execution of the next instruction. The first word written to the state frame is the format word, which specifies the size of the frame and the revision number of the FPCP. The MPU initiates the FSAVE instruction by reading the FPCP save CIR, which is encoded with a format word that indicates the appropriate action to be taken by the main processor. The current implementation of the FPCP always returns one of five responses in the save CIR:

Value	Definition
\$0018	Save NULL state frame
\$0118	Not ready, come again
\$0218	Illegal, take format exception
\$XX18	Save IDLE state frame
\$XXB4	Save BUSY state frame

where:
XX is the FPCP version number.

The Not Ready format word indicates that the FPCP is not prepared to perform a state save and that the MPU should process interrupts, if necessary, and the re-read the save CIR. The FPCP uses this format word to cause the main processor to wait while an internal operation is completed, if possible, in order to allow an IDLE frame to be saved rather than a BUSY frame. The Illegal format word is used to abort an FSAVE instruction that is attempted while the FPCP is executing a previous FSAVE instruction. All other format words cause the MPU to save the indicated state frame at the specified address. For state frame details see **State Frames** in the appropriate user's manual.

The following frame descriptions apply to both the **MC68040**, and the **MC68881/MC68882**.

- 4**
- NULL:** This state frame is four bytes long. An FSAVE instruction that generates this state frame indicates that the FPU state has not been modified since the last hardware reset or FRESTORE instruction with a NULL state frame. This indicates that the programmer's model is in the reset state, with nonsignaling NaNs in the floating-point data registers and zeros in the FPCR, FPSR, and FPIAR. (Thus, it is not necessary to save the programmer's model.)
- IDLE:** This state frame is four bytes long in the MC68040, 28 (\$1C) bytes long in the MC68881 and 60 (\$3C) bytes long in the MC68882. An FSAVE instruction that generates this state frame indicates that the FPU finished in an idle condition, waiting for the initiation of the next instruction, without any pending exceptions.
- BUSY:** This state frame is 96 (\$60) bytes long in the MC68040, 184 (\$B8) bytes long in the MC68881 and 216 (\$D8) bytes long in the MC68882. An FSAVE instruction that generates this size state frame indicates that the FPU encountered an exception in attempting to complete execution of the previous floating-point instructions.

The FSAVE does not save the programmer's model registers of the FPU, but is used to save only the user invisible portion of the machine. The FSAVE instruction may be used with the FMOVEM instruction to perform a full context save of the FPU including the floating-point data registers and system control registers. In order to accomplish a complete context save, an FSAVE instruction

FSAVE

Save Internal Floating-Point State
(MC68040/MC68881/MC68882)

FSAVE

is first executed to suspend the current operation and save the internal state, followed by the appropriate FMOVEM instructions to store the programmer’s model.

Floating Point Status Register: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					

Instruction Fields:

Effective Address field — Determines the addressing mode for the state frame. Only predecrement or control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

MOVE from SR

Move from the Status Register
(MC68010/MC68020/MC68030/MC68040/CPU32)

MOVE from SR

Operation: If supervisor state
then SR → Destination
else TRAP

Assembler

Syntax: MOVE SR,<ea>

Attributes: Size = (Word)

Description: Moves the data in the status register to the destination location. The destination is word length. Unimplemented bits are read as zeros.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	EFFECTIVE ADDRESS					
										MODE	REGISTER				

MOVE from SR

Move from the Status Register
(MC68010/MC68020/MC68030/MC68040/CPU32)

MOVE from SR

Instruction Fields:

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Available for the CPU32.

NOTE

Use the MOVE from CCR instruction to access only the condition codes.

MOVE to SR

Move to the Status Register
(M68000 Family)

MOVE to SR

Operation: If supervisor state
then Source \rightarrow SR
else TRAP

Assembler Syntax: MOVE <ea>,SR

Attributes: Size = (Word)

Description: Moves the data in the source operand to the status register. The source operand is a word and all implemented bits of the status register are affected.

Condition Codes:
Set according to the source operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

MOVE
to SR

Move to the Status Register
(M68000 Family)

MOVE
to SR

Instruction Fields:
Effective Address field — Specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*Available for the CPU32.

MOVE USP

Move User Stack Pointer
(M68000 Family)

MOVE USP

Operation: If supervisor state
then USP ∇ An or An ∇ USP
else TRAP

Assembler MOVE USP,An

Syntax: MOVE An,USP

Attributes: Size = (Long)

Description: Moves the contents of the user stack pointer to or from the specified address register.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	REGISTER		

Instruction Fields:

dr field — Specifies the direction of transfer:

0 — Transfer the address register to the USP.

1 — Transfer the USP to the address register.

Register field — Specifies the address register for the operation.

MOVEC

Move Control Register

(MC68010/MC68020/MC68030/MC68040/CPU32)

MOVEC

Operation: If supervisor state
then Rc \nrightarrow Rn or Rn \nrightarrow Rc
else TRAP

Assembler MOVEC Rc,Rn
Syntax: MOVEC Rn,Rc

Attributes: Size = (Long)

Description: Moves the contents of the specified control register (Rc) to the specified general register (Rn) or copies the contents of the specified general register to the specified control register. This is always a 32-bit transfer even though the control register may be implemented with fewer bits. Unimplemented bits are read as zeros.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D		REGISTER			CONTROL REGISTER										

Instruction Fields:

dr field — Specifies the direction of the transfer:

- 0 — Control register to general register.
- 1 — General register to control register.

A/D field — Specifies the type of general register:

- 0 — Data register.
- 1 — Address register.

Register field — Specifies the register number.
Control Register field — Specifies the control register.

Hex	Control Register
MC68010/MC68020/MC68030/MC68040/CPU32	
000	Source Function Code (SFC)
001	Destination Function Code (DFC)
800	User Stack Pointer (USP)
801	Vector Base Register (VBR)
MC68020/MC68030/MC68040	
002	Cache Control Register (CACR)
802	Cache Address Register (CAAR)*
803	Master Stack Pointer (MSP)
804	Interrupt Stack Pointer (ISP)
MC68040	
003	MMU Translation Control Register (TC)
004	Instruction Transparent Translation Register 0 (ITT0)
005	Instruction Transparent Translation Register 1 (ITT1)
006	Data Transparent Translation Register 0 (DTT0)
007	Data Transparent Translation Register 1 (DTT1)
805	MMU Status Register (MMUSR)
806	User Root Pointer (URP)
807	Supervisor Root Pointer (SRP)

Any other code causes an illegal instruction exception.
*For the MC68020 and MC68030 only.

MOVES

Move Address Space (MC68010/MC68020/MC68030/MC68040/CPU32)

MOVES

Operation: If supervisor state
then Rn → Destination [DFC] or Source [SFC] → Rn
else TRAP

Assembler MOVES Rn,<ea>
Syntax: MOVES <ea>,Rn

Attributes: Size = (Byte, Word, Long)

Description: Moves the byte, word, or long operand from the specified general register to a location within the address space specified by the destination function code (DFC) register; or, moves the byte, word, or long operand from a location within the address space specified by the source function code (SFC) register to the specified general register.

If the destination is a data register, the source operand replaces the corresponding low-order bits of that data register, depending on the size of the operation. If the destination is an address register, the source operand is sign extended to 32 bits and then loaded into that address register.

Condition Codes:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE		REGISTER			
A/D	REGISTER			dr	0	0	0	0	0	0	0	0	0		

MOVES

Move Address Space
(MC68010/MC68020/MC68030/MC68040/CPU32)

MOVES

Instruction Fields:

Size field — Specifies the size of the operation:

- 00 — Byte operation.
- 01 — Word operation.
- 10 — Long operation.

Effective Address field — Specifies the source or destination location within the alternate address space. Only memory alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—

MC68020, MC68030, AND MC68040 ONLY

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Available for the CPU32.

A/D field — Specifies the type of general register:

- 0 — Data register.
- 1 — Address register.

Register field — Specifies the register number.

dr field — Specifies the direction of the transfer:

- 0 — From (ea) to general register.
- 1 — From general register to (ea).

NOTE

For either of the two following examples with the same address register as both source and destination

MOVES.x An,(An) +

MOVES.x An, – (An)

the value stored is undefined. The current implementations of the MC68010, MC68020, MC68030, and MC68040 store the incremented or decremented value of An. Check the following code sequence to determine what value is stored for each case.

MOVEA.L #\$1000,A0

MOVES.L A0,(A0) +

MOVES.L A0, – (A0)

ORI to SR

Inclusive-OR Immediate to the Status Register
(M68000 Family)

ORI to SR

Operation: If supervisor state
then Source V SR ∇ SR
else TRAP

Assembler

Syntax: ORI #<data>,SR

Attributes: Size = (Word)

Description: Performs an inclusive-OR operation of the immediate operand and the contents of the status register and stores the result in the status register. All implemented bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set if bit 4 of immediate operand is one. Unchanged otherwise.
- N Set if bit 3 of immediate operand is one. Unchanged otherwise.
- Z Set if bit 2 of immediate operand is one. Unchanged otherwise.
- V Set if bit 1 of immediate operand is one. Unchanged otherwise.
- C Set if bit 0 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
WORD DATA (16 BITS)															

Operation: If Supervisor state
then if cc true
then PC + d \blacktriangleright PC
else trap

Assembler

Syntax: PBcc.<size><label>

Attributes: Size = (Word, Long)

Description: If the specified PMMU condition is met, execution continues at location (PC) + displacement. The displacement is a two’s complement integer which counts the relative distance in bytes. The value in the PC is the address of the displacement word(s). The displacement may be either 16 or 32 bits.

The condition specifier “cc” may specify the following conditions:

BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

BC	B clear	000001
LC	L clear	000011
SC	S clear	000101
AC	A clear	000111
WC	W clear	001001
IC	I clear	001011
GC	G clear	001101
CC	C clear	001111

PSR: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	1	SIZE	MC68851 CONDITION					
16-BIT DISPLACEMENT, OR MOST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT															
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)															

Instruction Fields:

- Size field — specifies the size of the displacement.
- 0 — The displacement is 16 bits.
 - 1 — The displacement is 32 bits.
- MC68851 Condition field — Specifies the coprocessor condition to be tested. This field is passed to the MC68851, which provides directives to the main processor for processing this instruction.
- Word Displacement field — The shortest displacement form for MC68851 branches is 16 bits.
- Long Word Displacement field — Allows a displacement larger than 16 bits.

PDBcc

Test, Decrement, and Branch
(MC68851)

PDBcc

Operation: If supervisor state
then If cc false
then (Dn-1 ∇ Dn; If Dn() -1 then PC + d \geq PC)
else no operation
else trap

Assembler
Syntax: PDBcc Dn, (label)

Attributes: Size = (Word)

Description: This instruction is a looping primitive of three parameters: an MC68851 condition, a counter (an MC68020 data register), and a 16-bit displacement. The instruction first tests the condition to determine if the termination condition for the loop has been met, and if so, the main processor proceeds to execute the next instruction in the instruction stream. If the termination condition is not true, the low order 16 bits of the counter register are decremented by one. If the result is not -1, execution continues at the location specified by the current value of the PC plus the sign extended 16-bit displacement. The value of the PC used in the branch address calculation is the address of the PDBcc instruction plus two.

4

The condition specifier “cc” may specify the following conditions:

BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

BC	B clear	000001
LC	L clear	000011
SC	S clear	000101
AC	A clear	000111
WC	W clear	001001
IC	I clear	001011
GC	G clear	001101
CC	C clear	001111

PSR: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	0	0	1	COUNT REGISTER		
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION					
16-BIT DISPLACEMENT															

Instruction Fields:

Register field — Specifies the data register in the main processor to be used as the counter.

MC68851 Condition field — Specifies the MC68851 condition to be tested. This field is passed to the MC68851, which provides directives to the main processor for processing this instruction.

Displacement field — Specifies the distance of the branch (in bytes).

Operation: If supervisor state
then invalidate ATC entries for Destination Addresses
else TRAP

Assembler PFLUSHA

Syntax: PFLUSH <fc>,#<mask>
PFLUSH <fc>,#<mask>,<ea>

Attributes: Unsized

Description: Invalidates ATC entries. The instruction has three forms. The PFLUSHA instruction invalidates all entries. When the instruction specifies a function code <fc> and mask <mask>, the instruction invalidates all entries for a selected function code or codes. When the instruction also specifies an effective address <ea>, the instruction invalidates the page descriptor for that effective address entry in each selected function code.

The <mask> operand contains three bits that correspond to the three function code bits. Each bit in the mask that is set to one indicates that the corresponding bit of the <fc> operand applies to the operation. Each bit in the mask that is zero indicates a bit of <fc> and of the function code that is ignored. For example, a mask operand of 100 causes the instruction to consider only the most significant bit of the <fc> operand. If the <fc> operand is 001, function codes 000, 001, 010, and 011 are selected.

The <fc> operand is specified in one of the following ways:

1. Immediate — Three bits in the command word.
2. Data Register — The three least significant bits of the data register specified in the instruction.
3. Source Function Code Register.
4. Destination Function Code Register.

Condition Codes:
Not affected.

MMUSR:
Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	1	MODE			0	0	MASK			FC				

Instruction Fields:

Effective Address field — Specifies a control alterable address. The ATC entry for this address is invalidated. Valid addressing modes are:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

NOTE

The address field must provide the memory management unit (MMU) with the effective address to be flushed from the ATC, not the effective address describing where the PFLUSH operand is located. For example, in order to flush the ATC entry corresponding to a logical address that is temporarily stored on top of the system stack, the instruction 'PFLUSH [(SP)]' must be used since 'PFLUSH (SP)' would invalidate the ATC entry mapping the system stack (i.e., the effective address passed to the MMU is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

Mode field — Specifies the type of flush operation:

001 — Flush all entries.

100 — Flush by function code only.

110 — Flush by function code and effective address.

Mask field — Mask for selecting function codes. Ones in the mask correspond to applicable bits; zeros are bits to be ignored. When mode is 001, mask must be 000.

FC field — Function code of entries to be flushed. When mode is 001, FC must be 00000.

10XXX — Function code is specified as bits XXX.

01DDD — Function code is specified as bits 2:0 of data register DDD.

00000 — Function code is specified as SFC register.

00001 — Function code is specified as DFC register.

PFLUSH

Flush ATC Entries
(MC68040)

PFLUSH

Operation: If supervisor state
then invalidate instruction and data ATC entries for destination
address
else TRAP

Assembler PFLUSH (An)
Syntax: PFLUSHN (An)
PFLUSHA
PFLUSHAN

Attributes: Unsized

Description: Invalidates ATC entries in both the instruction and data ATCs. The instruction has two forms. The PFLUSHA instruction invalidates all entries. The PFLUSH (An) instruction invalidates the entry in each ATC which matches the logical address in An and the specified function code.

4

The function code for PFLUSH is specified in the destination function code register. DFC values of 1 or 2 will result in flushing of user ATC entries in both ATCs, while values of 5 or 6 will result in flushing of supervisor ATC entries. PFLUSH is undefined for DFC values of 0, 3, 4, and 7 and may cause flushing of an unexpected entry.

The PFLUSHN and PFLUSHAN instructions have a global option specified, and invalidate only nonglobal entries. For example, if only page descriptors for operating system code have the global bit set, these two PFLUSH variants can be used to flush only user ATC entries during task swaps.

Condition Codes:
Not affected.

PFLUSH

Flush ATC Entries (MC68040)

PFLUSH

Instruction Format (Postincrement source and destination):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	0	0	0	OPMODE	REGISTER			

Instruction Fields:

Opmode field — Specifies the Flush Operation:

	Operation	Assembler Syntax
00	Flush page entry if not global	PFLUSHN (An)
01	Flush page entry	PFLUSH (An)
10	Flush all except global entries	PFLUSHAN
11	Flush all entries	PFLUSHA

Register field — Specifies the address register containing the effective address to be flushed when flushing a page entry.

PFLUSH PFLUSHA PFLUSHS

Invalidate Entries in the ATC
(MC68851)

PFLUSH PFLUSHA PFLUSHS

Operation: If Supervisor state
then ATC Entries for Destination Address are Invalidated
else trap

Assembler PFLUSHA
Syntax: PFLUSH <fc>,<#>,<mask>
PFLUSHS <fc>,<#>,<mask>
PFLUSH <fc>,<#>,<mask>,<ea>
PFLUSHS <fc>,<#>,<mask>,<ea>

Attributes: Unsigned

Description: PFLUSHA invalidates all entries in the ATC.

4

PFLUSH invalidates a set of ATC entries whose function code bits satisfy the relation: (ATC function code bits and <mask>) = (<fc> and <mask>) for all entries whose task alias matches the task alias currently active when the instruction is executed. With an additional effective address argument, PFLUSH invalidates a set of ATC entries whose function code satisfies the relation above, and whose effective address field matches the corresponding bits of the evaluated effective address argument. In both of these cases, ATC entries whose SG bit is set will not be invalidated unless the PFLUSHS is specified.

The function code for this operation may be specified to be:

1. Immediate — the function code is specified as four bits in the command word.
2. Data Register — the function code is contained in the lower four bits in the MC68020 data register specified in the instruction.
3. Source Function Code Register — the function code is contained in the source function code (SFC) register in the CPU. Since the SFC of the MC68020 has only three implemented bits, only function codes \$0 through \$7 can be specified in this manner.

PFLUSH
PFLUSHA
PFLUSHS

Invalidate Entries in the ATC
(MC68851)

PFLUSH
PFLUSHA
PFLUSHS

4. Destination Function Code Register — the function code is contained in the destination function code (DFC) register in the CPU. Since the DFC of the MC68020 has only three implemented bits, only function codes \$0 through \$7 can be specified in this manner.

PSR: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	1	MODE			0	MASK			FC					

Instruction Fields:

Effective Address field — Specifies an address whose page descriptor is to be flushed from (invalidated) the ATC. Only control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

PFLUSH PFLUSHA PFLUSHS

Invalidate Entries in the ATC
(MC68851)

PFLUSH PFLUSHA PFLUSHS

Note that the effective address field must provide the MC68851 with the effective address of the entry to be flushed from the ATC, not the effective address describing where the PFLUSH operand is located. For example, in order to flush the ATC entry corresponding to a logical address that is temporarily stored on the top of the system stack, the instruction 'PFLUSH [(SP)]' must be used since 'PFLUSH (SP)' would invalidate the ATC entry mapping the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

Mode field — Specifies how the ATC is to be flushed.

- 001 — Flush all entries
- 100 — Flush by function code only
- 101 — Flush by function code including shared entries
- 110 — Flush by function code and effective address
- 111 — Flush by function code and effective address including shared entries

Mask field — Indicates which bits are significant in the function code compare.

A zero indicates that the bit position is not significant, a one indicates that the bit position is significant. If mode = 001 (flush all entries), mask must be 0000.

FC field — Function code of address to be flushed. If mode = 001 (flush all entries), function code must be 00000

Otherwise:

- 1DDDD — Function code is specified as four bits DDDD
- 01RRR — Function code is contained in CPU data register RRR
- 00000 — Function code is contained in CPU SFC register
- 00001 — Function code is contained in CPU DFC register

PFLUSHR

Invalidate ATC and RPT Entries
(MC68851)

PFLUSHR

Operation: If Supervisor state
then the RPT entry (if any) matching the root pointer specified by
<ea> and corresponding ATC entries are invalidated
else trap

Assembler

Syntax: PFLUSHR<ea>

Attributes: Unsized

Description: The double long word pointed to by <ea> is taken to be a previously used value of the CRP register. The RPT entry matching this CRP (if any) is flushed, and all ATC entries loaded with this value of CRP (except for those that are globally shared) are invalidated. If no entry in the RPT matches the operand of this instruction, then no action is taken.

If the supervisor root pointer is not in use, the operating system should not issue the PFLUSHR command to destroy a task identified by the current CRP. It should wait until the CRP has been loaded with the root pointer identifying the next task until using the PFLUSHR instruction. At any time, execution of the PFLUSHR instruction for the current CRP causes the current task alias to be corrupted.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field — Specifies the address of a previous value of the CRP register. Only memory addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Note that the effective address usage of this instruction is different than that of other PFLUSH variants.

PLOAD

Load an Entry into the ATC (MC68030/MC68851)

PLOAD

Operation: If Supervisor state
 then search translation table and make ATC entry for effective
 address
 else trap

Assembler PLOADR <function code>,<ea>

Syntax: PLOADW <function code>,<ea>

Attributes: Unsized

Description: For the **MC68851** PLOAD the translation table is searched for a translation for the specified effective address. If one is found, it is flushed from the ATC, and an entry is made in the ATC as if a bus master had run a bus cycle. Used and modified bits in the table are updated as part of the table search. The MC68851 ignores the logical bus arbitration signals during the flush and load phase at the end of this instruction preventing the possibility of an entry temporarily disappearing from the ATC and causing a spurious table search.

This instruction will cause a PMMU illegal operation exception (vector \$39) if the E bit of the TC register is clear.

The function code for this operation may be specified to be:

1. Immediate — the function code is specified as four bits in the command word.
2. Data Register — the function code is contained in the lower four bits in the MC68020 data register specified in the instruction.
3. Source Function Code — the function code is contained in the source function code (SFC) register in the CPU. Since the SFC of the MC68020 has only three implemented bits, only function codes \$0 through \$7 can be specified in this manner.
4. Destination Function Code Register — the function code is contained in the destination function code (DFC) register in the CPU. Since the DFC of the MC68020 has only three implemented bits, only function codes \$0 through \$7 can be specified in this manner.

PLOAD

Load an Entry into the ATC (MC68030/MC68851)

PLOAD

For the **MC68030** PLOAD searches the ATC for the specified effective address. Also searches the translation table for the descriptor corresponding to the specified effective address. A new entry is created as if the MC68030 had attempted to access that address. Sets the used and modified bits appropriately as part of the search. The instruction executes regardless of the value of the E bit in the translation control (TC) register or the state of the MMUDIS signal.

The (function code) operand is specified in one of the following ways:

1. Immediate — Three bits in the command word.
2. Data Register — The three least significant bits of the data register specified in the instruction.
3. Source Function Code Register.
4. Destination Function Code Register.

The effective address field specifies the logical address whose translation is to be loaded.

PLOADR causes U bits in the translation tables to be updated as if a read access had taken place. PLOADW causes U and M bits in the translation tables to be updated as if a write access had taken place.

PSR: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	1	0	0	0	R/W	0	0	0	0	FC				

PLOAD

Load an Entry into the ATC (MC68030/MC68851)

PLOAD

Instruction Fields:

Effective Address field — Specifies the logical address whose translation is to be loaded into the ATC. Only control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Note that the effective address field must provide the MC68851 with the effective address of the entry to be loaded into the ATC, not the effective address describing where the PLOAD operand is located. For example, in order to load an ATC entry to map a logical address that is temporarily stored on the system stack, the instruction PLOAD [(SP)] must be used since PLOAD (SP) would load an ATC entry mapping the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

R/W field — Specifies whether the tables should be updated for a read or a write

- 1 — Read
- 0 — Write

FC field — Function code of address to load (**MC68851**)

1DDDD — Function code is specified as four bits DDDD

01RRR — Function code is contained in CPU data register RRR

00000 — Function code is contained in CPU SFC register

00001 — Function code is contained in CPU DFC register

FC field — Function code of address corresponding to entry to be loaded
(**MC68030**).

10XXX — Function code is specified as bits XXX.

01DDD — Function code is specified as bits 2:0 of data register DDD.

00000 — Function code is specified as SFC register.

00001 — Function code is specified as DFC register.

Operation: If supervisor state
then (Source) ∇ MRn or MRn ∇ (Destination)

Assembler PMOVE MRn,<ea>

Syntax: PMOVE <ea>,MRn
PMOVEFD <ea>,MRn

Attributes: Size = (Word, Long, Quad)

Description: Moves the contents of the source effective address to the specified MMU register, or moves the contents of the MMU register to the destination effective address.

The instruction is a quad word (8 byte) operation for the CPU root pointer (CRP) and the supervisor root pointer (SRP). It is a long word operation for the translation control register (TC) and the transparent translation registers (TT0 and TT1). It is a word operation for the MMU status register (MMUSR).

The PMOVEFD (PMOVE with flush disable) form of this instruction sets the FD bit to disable flushing the ATC when a new value is loaded into the SRP, CRP, TT0, TT1 or TC register (but not the MMUSR).

Writing to the following registers has the indicated side effects.

CPU root pointer

When the FD bit is zero, flushes the ATC. If the operand value is invalid for a root pointer descriptor, the instruction takes an MMU configuration error exception after moving the operand to the CRP.

Supervisor root pointer

When the FD bit is zero, flushes the ATC. If the value of the operand is invalid as a root pointer descriptor, the instruction takes an MMU configuration error exception after moving the operand to the SRP.

Translation control

When the FD bit is zero, flushes the ATC. Consistency checks are performed on the PS and Tlx fields, if the E bit value is one. If check fails, the instruction takes an MMU configuration exception after moving the operand to the TC. If the checks passes, the TC register is loaded with the operand with E clear.

Transparent translation

When the FD bit is zero, flushes the ATC. Enables or disables the transparent translation register according to the E bit written. If the E bit is set to one, the transparent translation register is enabled. If the E bit is zero, the register is disabled.

Condition Codes:

Not affected.

MMUSR:

Not affected (unless the MMUSR is specified as the destination operand).

Instruction Format (for SRP, CRP, and TC registers):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	1	0	P REG			R/W	FD	0	0	0	0	0	0	0	0

Instruction Fields (for SRP, CRP, and TC registers):

Effective Address field — Specifies the memory location for the transfer. Only control alterable addressing modes are allowed.

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	—	—
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

- P Reg field — Specifies the MMU register:
000 — TC
010 — SRP
011 — CRP
- R/W field — Specifies the direction of transfer:
0 — Memory to MMU register
1 — MMU register to memory
- FD field — Disables flushing of the ATC on writes to MMU registers:
0 — ATC is flushed
1 — ATC is not flushed

Instruction Format (for MMUSR):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	0	0

Instruction Fields (for MMUSR):

- Effective Address field — Specifies the memory location for the transfer. Control alterable addressing modes shown for SRP register apply.
- R/W field — Specifies the direction of transfer:
0 — Memory to MMU status register
1 — MMU status register to memory

NOTE

The syntax of assemblers for the MC68851 use the symbol PSR for the MMU status register.

Instruction Format (for TT registers):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	0	P REG			R/W	FD	0	0	0	0	0	0	0	0

Instruction Fields (for TT registers):

Effective Address field — Specifies the memory location for the transfer. Control alterable addressing modes shown for SRP register apply.

P Reg field — Specifies the TT register:

010 — Transparent translation register 0.

011 — Transparent translation register 1.

R/W field — Specifies the direction of transfer:

0 — Memory to MMU status register

1 — MMU status register to memory

FD field — Disables flushing of the ATC:

0 — ATC is flushed

1 — ATC is not flushed

PMOVE

Move PMMU Register (MC68851)

PMOVE

Operation: If Supervisor state
then MC68851 Register ♦ Destination or Source ♦ MC68851
Register
else trap

Assembler PMOVE <PMMU Register>,<ea>

Syntax: PMOVE <ea>,<PMMU Register>

Attributes: Size = (Byte, Word, Long, Double Long)

Description: The contents of the MC68851 register is copied to the address specified by <ea>, or the data at <ea> is copied into the MC68851 register.

PMOVE is a double long (eight byte) operation for the following registers: CRP, SRP, DRP.

PMOVE is a long (four byte) operation for the following register: TC.

PMOVE is a word (two byte) operation for the following registers: BAC, BAD, AC, PSR, PCSR.

PMOVE is a byte (one byte) operation for the following registers: CAL, VAL, SCC.

This instruction has side effects when data is read into certain registers. These effects are:

- CRP —Causes the internal root pointer table to be searched for the new value. If a matching value is not found, an entry in the root pointer table is selected for replacement, and all ATC entries associated with the replaced entry are invalidated.
- SRP —Cause all entries in the ATC that were formed with the SRP (even globally shared entries) to be invalidated.
- DRP —Causes all entries in the ATC that were formed with the DRP (even globally shared entries) to be invalidated.
- TC —If data written to the TC register attempts to set the E bit (and the E bit is currently clear), a consistency check is performed on the IS, TIA, TIB, TIC, TID, and PS fields.

PSR: Not affected unless the PSR is written to by the instruction.

Instruction Format 1 (PMOVE to/from TC, CRP, DRP, SRP, CAL, VAL, SCC, AC):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	0	P REG			R/W	0	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field — for memory-to-register transfers, any addressing mode is allowed as shown:

Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
{xxx}.W	111	000
{xxx}.L	111	001
#<data>	111	100
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*PMOVE to CRP, SRP, DRP not allowed with these modes.

For register-to-memory transfers, only alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn*	000	reg. number:Dn	(xxx).W	111	000
An*	001	reg. number:An	(xxx).L	111	001
(An)	010	reg. number:An	#(data)	—	—
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	—	—
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

*PMOVE from CRP, SRP,DRP not allowed with these modes.

Register field — Specifies the MC68851 register

- 000 — TC
- 001 — DRP
- 010 — SRP
- 011 — CRP
- 100 — CAL
- 101 — VAL
- 110 — SCC
- 111 — AC

R/W field — Specifies the direction of transfer

- 0 — Transfer (ea) to MC68851 register
- 1 — Transfer MC68851 register to (ea)

Instruction Format 2 (PMOVE to/from BADx, BACx):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	1	1	P REG			R/W	0	0	0	0	NUM			0	0

Instruction Fields:

Effective Address field — Same as above

P Register field — Specifies the type of MC68851 register

100 — BAD

101 — BAC

R/W field — Specifies the direction of transfer

0 — Transfer (ea) to MC68851 register

1 — Transfer MC68851 register to (ea)

Num field — Specifies the number of the BACx or BADx register to be used

Instruction Format 3 (PMOVE to/from PSR, from PCSR):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	1	P REG			R/W	0	0	0	0	0	0	0	0	0

Instruction Fields:

Effective Address field — Same as above

P Register field — Specifies the MC68851 register

000 — PSR

001 — PCSR

R/W field — Specifies direction of transfer

0 — Transfer (ea) to MC68851 register

1 — Transfer MC68851 register to (ea) (must be one to access PCSR using this format)

PRESTORE

PMMU Restore Function (MC68851)

PRESTORE

Operation: If Supervisor state
then MC68851 State Frame → Internal State, Programmer Registers
else trap

Assembler

Syntax: PRESTORE (ea)

Attributes: Unsized, Privileged

Description: The MC68851 aborts execution of any operation it was performing, and a new internal state and programmer registers are loaded from the state frame located at the effective address. The first word at the specified address is the format word of the state frame, which specifies the size of the frame and the revision number of the MC68851 that created it. The MC68020 will write the first word to the MC68851 restore coprocessor interface register to initiate the restore operation and then read the response coprocessor interface register to verify that the MC68851 recognizes the format as valid. If the format word is invalid for this MC68851 (either because the size of the frame is not recognized, or the revision number does not match the revision of this MC68851), then the MC68020 is instructed to take a format exception, and the MC68851 returns to the idle state with its user visible registers unchanged. If the format word is valid, the appropriate state frame is loaded, starting at the specified location and up through higher addresses.

The PRESTORE instruction restores the nonuser visible state of the MC68851 as well as the PSR, CRP, SRP, CAL, VAL, and SCC registers of the user programming model. In addition, if any breakpoints are enabled, all BACx and BADx registers are restored.

This instruction is the inverse of the PSAVE instruction.

The current implementation of the MC68851 supports four state sizes.

NULL: This state frame is four bytes long, with a format word of \$0. A PRESTORE with this size state frame places the MC68851 in the idle state with no coprocessor or module operations in progress.

IDLE: This state frame is 36 (\$24) bytes long. A PRESTORE with this size state frame causes the MC68851 to place itself in an idle state with no coprocessor operations in progress, and no breakpoints enabled. A module operation may or may not be in progress. The minimal set of MC68851 registers are restored by this state frame.

MID-COPROCESSOR: This state frame is 44 (\$2C) bytes long. A PRESTORE with this size frame restores the MC68851 to a state with a coprocessor operation in progress, and no breakpoints enabled.

BREAKPOINTS ENABLED: This state frame is 76 (\$4C) bytes long. A PRESTORE with this size state frame restores all of the breakpoint registers, along with other state. A coprocessor operation may or may not be in progress.

PSR: Set according to restored data.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	1	0	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

Effective Address field — Specifies the source location. Only control or post-increment addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
-(An)	—	—
(d16,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
((bd,An,Xn),od)	110	reg. number:An
((bd,An),Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d16,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011
((bd,PC,Xn),od)	111	011
((bd,PC),Xn,od)	111	011

Operation: If Supervisor state
then MC68851 Internal State, Programmer Registers → State Frame
else trap

Assembler

Syntax: PSAVE <ea>

Attributes: Unsized, Privileged

Description: The MC68851 suspends execution of any operation that it was performing and saves its internal state and certain programmer registers in a state frame located at the effective address. The registers copied are: PSR, CRP, SRP, CAL, VAL, and SCC. In addition, if any breakpoints are enabled, all BAC and BAD registers are copied. After the save operation, the MC68851 is in an idle state waiting for another operation to be requested. Programmer registers are not changed.

The state frame format saved by the MC68851 depends on its state at the time of the PSAVE operation. In the current implementation, three format frames are possible.

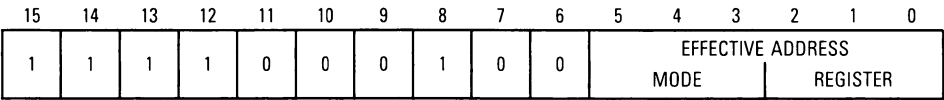
IDLE: This state frame is 36 (\$24) bytes long. A PSAVE of this size state frame indicates that the MC68851 was in an idle state with no coprocessor operations in progress, and no breakpoints enabled. A module call operation may or may not have been in progress when this state frame was saved.

MID-COPROCESSOR: This state frame is 44 (\$2C) bytes long. A PSAVE of this size frame indicates that the MC68851 was in a state with a coprocessor or module call operation in progress, and no breakpoints enabled.

BREAKPOINTS ENABLED: This state frame is 76 (\$4C) bytes long. A PSAVE of this size state frame indicates that one or more breakpoints were enabled. A coprocessor or module call operation may or may not have been in progress.

PSR: Not affected

Instruction Format:



Instruction Fields:

Effective Address field — Specifies the destination location. Only control or predecrement addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	—	—
(dg,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Operation: If Supervisor state
then if cc true
then 1s ➡ Destination
else 0s ➡ Destination
else trap

Assembler

Syntax: PScc <ea>

Attributes: Size = (Byte)

Description: The specified MC68851 condition code is tested. If the condition is true, the byte specified by the effective address is set to TRUE (all ones); otherwise, that byte is set to FALSE (all zeros).

The condition code specifier “cc” may specify the following conditions:

BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

BC	B clear	000001
LC	L clear	000011
SC	S clear	000101
AC	A clear	000111
WC	W clear	001001
IC	I clear	001011
GC	G clear	001101
CC	C clear	001111

PSR: Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	EFFECTIVE ADDRESS					
										MODE	REGISTER				
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION					

Instruction Fields:

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	—	—
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d ₁₆ ,An)	101	reg. number:An	(d ₁₆ ,PC)	—	—
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

MC68851 Condition field — Specifies the coprocessor condition to be tested. This field is passed to the MC68851, which provides directives to the main processor for processing this instruction.

Operation: If supervisor state
then logical address status \blacktriangledown MMUSR
else TRAP

Assembler Syntax: PTESTR <function code>,<ea>,<#(level)>
PTESTR <function code>,<ea>,<#(level)>,An
PTESTW <function code>,<ea>,<#(level)>
PTESTW <function code>,<ea>,<#(level)>,An

Attributes: Unsized

Description: This instruction searches the ATC or the translation tables to a specified level for the translation descriptor corresponding to the <ea> field and sets the bits of the MMU status register (MMUSR) according to the status of the descriptor. Optionally, PTEST stores the physical address of the last table entry accessed during the search in the specified address register. The PTEST instruction searches the ATC or the translation tables to obtain status information, but alters neither the used or modified bits of the translation tables nor the ATC. When the level operand is zero, transparent translation of only either read or write accesses causes the operations of the PTESTR and PTESTW to return different results.

The <function code> operand is specified in one of the following ways:

1. Immediate — Three bits in the command word.
2. Data Register — The three least significant bits of the data register specified in the instruction.
3. Source Function Code Register.
4. Destination Function Code Register.

The effective address is the address to test. The <level> operand specifies the level of the search. Level 0 specifies searching the ATC only. Values 1–7 specify searching the translation tables only. The search ends at the specified level. A level 0 test does not return the same MMUSR values as a test at a nonzero level number.

Execution of the instruction continues to the requested level, or until one of the following conditions is detected:

- Invalid descriptor.
- Limit violation.
- Bus error assertion (physical bus error).

The instruction accumulates status as it accesses successive table entries.

When the instruction specifies an ATC search with an address register operand, the MC68030 takes an F-line unimplemented instruction exception.

If an address register parameter is specified for a translation table search, the physical address of the last descriptor successfully fetched is loaded into the address register. A descriptor is "successfully" fetched if, and only if, all portions of the descriptor can be read by the MC68030 without abnormal termination of the bus cycle. If the DT field of the root pointer used indicates "page descriptor", the returned address is \$0. For a long descriptor, the address of the first long word is returned. The size of the descriptor (short or long) is not returned, and must be determined from a knowledge of the translation table.

Condition Codes:
Not affected.

MMUSR:

B	L	S		W	I	M			T				N
*	*	*	0	*	*	*	0	0	0	0	0	0	*

The MMU status register contains the results of the search. The values in the fields of the MMUSR for an ATC search are:

MMUSR Bit	PTEST, Level 0	PTEST, Level 1–7
Bus Error (B)	This bit is set if the bus error bit is set in the ATC entry for the specified logical address.	This bit is set if a bus error is encountered during the table search for the PTEST instruction.
Limit (L)	This bit is cleared.	This bit is set if an index exceeds a limit during the table search.
Supervisor Violation (S)	This bit is cleared.	This bit is set if the S bit of a long (S) format table descriptor or long format page descriptor encountered during the search is set, and the FC2 bit of the function code specified by the PTEST instruction is not equal to one. The S bit is undefined if the I bit is set.
Write Protected (W)	The bit is set if the WP bit of the ATC entry is set. It is undefined if the I bit is set.	This bit is set if a descriptor or page descriptor is encountered with the WP bit set during the table search. The W bit is undefined if the I bit is set.
Invalid (I)	This bit indicates an invalid translation. The I bit is set if the translation for the specified logical address is not resident in the ATC, or if the B bit of the corresponding ATC entry is set.	This bit indicates an invalid translation. The I bit is set if the DT field of a table or a page descriptor encountered during the search is set to invalid, or if either the B or L bits of the MMUSR are set during the table search.
Modified (M)	This bit is set if the ATC entry corresponding to the specified address has the modified bit set. It is undefined if the I bit is set.	This bit is set if the page descriptor for the specified address has the modified bit set. It is undefined if I is set.
Transparent (T)	This bit is set if a match occurred in either (or both) of the transparent translation registers (TT0 or TT1).	This bit is set to zero.
Number of Levels (N)	This 3-bit field is cleared to zero.	This 3-bit field contains the actual number of tables accessed during the search.

Instruction Field:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
1	0	0	LEVEL			R/W	A	REG			FC				

Instruction Fields:

Effective Address field — Specifies the logical address to be tested. Only control
alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	—	—
(An) +	—	—			
– (An)	—	—			
(d16,An)	101	reg. number:An	(d16,PC)	—	—
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

Level field — Specifies the highest numbered level to be searched in the table.
When this field contains 0, the A field and the Reg field must also be 0. The
instruction takes an F-line exception when the level field is 0 and the A field
is not 0.

R/W field — Specifies simulating a read or write bus cycle (no difference for
MC68030 MMU):

- 0 — Write
- 1 — Read

A field — Specifies the address register option:

- 0 — No address register.
- 1 — Return the address of the last descriptor searched in the address register
specified in the Reg field.

Reg field — Specifies an address register for the instruction. When the A field contains 0, this field must contain 0.

FC field — Function code of address to be tested:

10XXX — Function code is specified as bits XXX.

01DDD — Function code is specified as bits 2:0 of data register DDD.

00000 — Function code is specified as SFC register.

00001 — Function code is specified as DFC register.

Operation: If supervisor state
 then logical address status ∇ MMUSR; entry ∇ ATC
 else TRAP

Assembler PTESTR (An)
Syntax: PTESTW (An)

Attributes: Unsized

Description: This instruction searches the translation tables for the page descriptor corresponding to the test address in An and sets the bits of the MMU status register (MMUSR) according to the status of the descriptors. The upper address bits of the translated physical address are also stored in the MMUSR. The PTESTR instruction simulates a read access and sets the U bit in each descriptor during table searches, while PTESTW simulates a write access and also sets the M bit in the descriptors, the ATC entry, and the MMUSR.

A matching entry in the ATC (data or instruction) specified by the function code will be flushed by PTEST. Completion of PTEST will result in the creation of a new ATC entry.

The function code for the test address is specified in the destination function code register. A PTEST instruction with a DFC value of 0, 3, 4, or 7 is undefined, and will return an unknown value in the MMUSR.

Execution of the instruction continues until one of the following conditions:

- Match with one of the two TTR registers
- Transfer error assertion (physical transfer error)
- Invalid descriptor
- Valid page descriptor

Condition Codes:
Not affected.

MMUSR:

PHYSICAL ADDRESS	B	G	U1	U0	S	CM	M	W	T	R	
*	*	*	*	*	*	*	*	0	*	*	*

PTEST

Test a Logical Address
(MC68040)

PTEST

The MMU status register contains the results of the search. The values in the fields of the MMUSR for a search are:

- Physical Address** This 20-bit field contains the upper bits of the translated physical address. The actual physical address is formed by merging these bits with the lower bits of the logical address.
- B** Bus Error. Set if a transfer error is encountered during the table search for the PTEST instruction. If B is set, all other bits are zero.
- G** Globally Shared. This bit is set if the G bit is set in the page descriptor.
- U1, U0** User Page Attributes. These bits are set if corresponding bits in the page descriptor are set.
- S** Supervisor Protection. This bit is set if the S bit in the page descriptor is set. This bit does not indicate that a violation has occurred.
- CM** Cache Mode. This 2-bit field is copied from the CM bits in the page descriptor.
- M** Modified. This bit is set if M bit is set in the page descriptor associated with the address.
- W** Write Protect. This bit is set if the W bit is set in any of the descriptors encountered during the table search. Setting of this bit does not indicate that a violation occurred.
- T** Transparent Translation Register Hit. If T is set, then the PTEST address matched an instruction or data TTR register, the R bit is set, and all other bits are zero.
- R** Resident. Set if the PTEST address matches a TTR register, or if the table search completes by obtaining a valid page descriptor.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	0	1	R/W	0	1	REGISTER		

Instruction Fields:

- R/W field — Specifies simulating a read or write bus transfer:
 - 0 — Write
 - 1 — Read
- Register field — Specifies the address register containing the effective address for the instruction.

Operation: If Supervisor state
then Information about Logical Address ♦ PSTATUS
else trap

Assembler PTESTR <function code>,<ea>,<#(level)>[,An]

Syntax: PTESTW <function code>,<ea>,<#(level)>[,An]

Attributes: Unsized

Description: If the E bit of the TC register is set, information about the logical address specified by <fc> and <ea> is placed in the PSR register. If the E bit of the TC register is clear this instruction will cause a PMMU Illegal Operation Exception (vector \$39).

The function code for this operation may be specified to be:

1. Immediate — the function code is specified as four bits in the command word.
2. Data Register — the function code is contained in the lower four bits in the MC68020 data register specified in the instruction.
3. Source Function Code Register — the function code is contained in the source function code (SFC) register in the CPU. Since the SFC of the MC68020 has only three implemented bits, only function codes \$0 through \$7 can be specified in this manner.
4. Destination Function Code Register — the function code is contained in the destination function code (DFC) register in the CPU. Since the DFC of the MC68020 has only three implemented bits, only function codes \$0 through \$7 can be specified in this manner.

The effective address field specifies the logical address to be tested.

The <level> parameter specifies the depth to which the translation table is to be searched. A value of zero specifies a search of the ATC only. Values one through seven cause the ATC to be ignored and specify the maximum number of descriptors to fetch. Note that finding an ATC entry with <level> set to zero may result in a different value in the PSR register than forcing a table search. Only the I, W, G, M, and C bits of the PSR register are always the same in both cases.

Either PTESTR or PTESTW must be specified. The two instructions differ in the setting of the A bit of the PSR. For systems where access levels are not in use, either PTESTR or PTESTW may be used. U and M bits in the translation table are not modified by this instruction.

If an address register parameter is specified, the physical address of the last descriptor successfully fetched is loaded into the address register. A descriptor is 'successfully' fetched if, and only if, all portions of the descriptor can be read by the MC68851 without abnormal termination of the bus cycle. If the DT field of the root pointer used indicates 'page descriptor', the returned address is \$0.

The PTEST instruction continues searching the translation tables until the requested level is reached or until a condition occurs that makes further searching impossible (i.e., a DT field set to 'invalid', a limit violation, or a bus error from memory). The information in the PSR register reflects the accumulated values.

PSR Register: Set as follows:

- B Set if a bus error was received during a descriptor fetch, or if, $\langle \text{level} \rangle = 0$ and an entry was found in the ATC with its BERR bit set. Cleared otherwise.
- L Set if the limit field of a long descriptor was exceeded. Cleared otherwise.
- S Set if a long descriptor indicated supervisor-only access and the $\langle \text{fc} \rangle$ parameter did not have bit [2] set. Cleared otherwise.
- A If PTESTR was specified, set if the RAL field of a long descriptor would deny access. If PTESTW was specified, set if a WAL or RAL field of a long descriptor would deny access. Cleared otherwise.
- W Set if the WP bit of a descriptor was set, or if a WAL field of a long descriptor would deny access.
- I Set if a valid translation was not available. Cleared otherwise.
- M If the tested address is found in the ATC, then set to the value of the M bit in the ATC. If the tested address is found in the translation table, set if the M bit of the page descriptor is set, and cleared otherwise.
- G If the tested address is found in the ATC, then set to the value of the G bit in the ATC. If the tested address is found in the translation table, set if the G bit of the page descriptor is set, and cleared otherwise.
- C Set if the address is globally shared. Cleared otherwise.

N Set to the number of levels searched. A value of zero indicates an early termination of the table search in the root pointer (DT = 'page descriptor') if the level specification was not zero. If the level specification was zero, N is always set to zero.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
1	0	0	LEVEL			R/W	A REG				FC				

Instruction Fields:

Effective Address field — Specifies the logical address about which information is requested. Only control alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#{data}	—	—
(An) +	—	—			
– (An)	—	—			
(d16,An)	101	reg. number:An	(d16,PC)	—	—
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

Note that the effective address field must provide the MC68851 with the effective address of the logical address to be tested, not the effective address describing where the PTEST operand is located. For example, in order to test a logical address that is temporarily stored on the system stack, the instruction PTEST [(SP)] must be used since PTEST (SP) would test the mapping of the system stack (i.e., the effective address passed to the MC68851 is the effective address of the system stack, not the effective address formed by the operand located on the top of the stack).

Level — Specifies the depth to which the translation table should be searched

R/ \overline{W} field — Specifies whether the A bit should be updated for a read or a write

1 — Read

0 — Write

Areg — Specifies the address register in which to load the last descriptor address

0xxx — Do not return the last descriptor address to an address register

1RRR — Return the last descriptor address to address register RRR

NOTE

When the PTEST instruction specifies a level of zero, the Areg field must be 0000. Otherwise, an F-line exception is generated.

FC field — Function code of address to test

1DDDD — Function code is specified as four bits DDDD

01RRR — Function code is contained in CPU data register RRR

00000 — Function code is contained in CPU SFC register

00001 — Function code is contained in CPU DFC register

PTRAPcc

Trap on PMMU Condition
(M68851)

PTRAPcc

Operation: If Supervisor state
then if cc true then trap
else trap

Assembler PTRAPcc
Syntax: PTRAPcc.W #<data>
PTRAPcc.L #<data>

Attributes: Unsize or Size = (Word, Long)

Description: If the selected MC68851 condition is true, the processor initiates exception processing. The vector number is generated to reference the cpTRAPcc exception vector, the stacked program counter is the address of the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction. The immediate data operand is placed in the next word(s) following the MC68851 condition and is available for user definition for use within the trap handler. Following the condition word may be a user-defined data operand specified as immediate data, to be used by the trap handler.

The condition specifier "cc" may specify the following conditions:

BS	B set	000000
LS	L set	000010
SS	S set	000100
AS	A set	000110
WS	W set	001000
IS	I set	001010
GS	G set	001100
CS	C set	001110

BC	B clear	000001
LC	L clear	000011
SC	S clear	000101
AC	A clear	000111
WC	W clear	001001
IC	I clear	001011
GC	G clear	001101
CC	C clear	001111

PSR: Not affected

PTRAPcc

Trap on PMMU Condition
(M68851)

PTRAPcc

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	1	1	1	OPMODE		
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION					
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)															
LEAST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)															

Instruction Fields:

- Opmode field — Selects the instruction form
- 010 — Instruction is followed by one operand word
- 011 — Instruction is followed by two opearnd words
- 100 — Instruction has no following operand words
- MC68851 Condition field — Specifies the coprocessor condition to be tested.
- This field is passed to the MC68851, which provides directives to the main processor for processing this instruction.

RESET

Reset External Devices (M68000 Family)

RESET

Operation: If supervisor state
then Assert $\overline{\text{RESET}}$ ($\overline{\text{RSTO}}$, MC68040 only) Line
else TRAP

Assembler

Syntax: RESET

Attributes: Unsized

Description: Asserts the $\overline{\text{RSTO}}$ signal for 512 (124 for MC68000, MC68008, and MC68010) clock periods, resetting all external devices. The processor state, other than the program counter, is unaffected and execution continues with the next instruction.

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Operation: If supervisor state
 then (SP) ∇ SR; SP + 2 ∇ SP; (SP) ∇ PC;
 SP + 4 ∇ SP;
 restore state and deallocate stack according to (SP)
 else TRAP

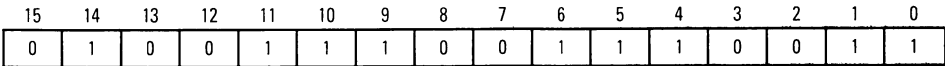
Assembler Syntax: RTE

Attributes: Unsized

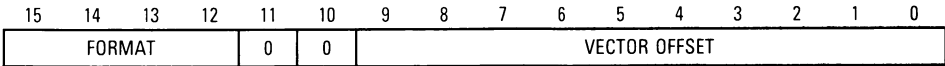
Description: Loads the processor state information stored in the exception stack frame located at the top of the stack into the processor. The instruction examines the stack format field in the format/offset word to determine how much information must be restored.

Condition Codes:
 Set according to the condition code bits in the status register value restored from the stack.

Instruction Format:



Format/Offset Word (in stack frame) (MC68010/MC68020/MC68030/MC68040/CPU32):



Format Field of Format/Offset Word:

- Contains the format code, which implies the stack frame size (including the format/offset word).
- 0000 — Short Format, removes four words. Loads the status register and the program counter from the stack frame.
 - 0001 — Throwaway Format, removes four words. Loads the status register from the stack frame and switches to the active system stack. Continues the instruction using the active system stack.

- 0010 — Instruction Error Format, removes six words. Loads the status register and the program counter from the stack frame and discards the other words.
- 0111 — Access Error Format, removes 30 words.
- 1000 — MC68010 Long Format, removes 29 words.
- 1001 — MC68020/MC68030 Mid-Instruction Format, removes 10 words.
- 1010 — MC68020/MC68030 Short Format, removes 16 word.
- 1011 — MC68020/MC68030 Long Format, removes 46 words.
- 1100 — CPU32 Bus Error Format, processor state is recovered from stack in addition to the SR and PC.

Format	Supported By				
	MC68010	MC68020	MC68030	MC68040	CPU32
0000	✓	✓	✓	✓	✓
0001	—	✓	✓	✓	—
0010	—	✓	✓	✓	✓
0111	—	—	—	✓	—
1000	✓	—	—	—	—
1001	—	✓	✓	—	—
1010	—	✓	✓	—	—
1011	—	✓	✓	—	—
1100	—	—	—	—	✓

NOTE: An unsupported format causes the processor to take a format error exception.

Operation: If supervisor state
then Immediate Data ∇ SR; STOP
else TRAP

Assembler

Syntax: STOP #<data>

Attributes: Unsized

Description: Moves the immediate operand into the status register (both user and supervisor portions), advances the program counter to point to the next instruction, and stops the fetching and executing of instructions. A trace, interrupt, or reset exception causes the processor to resume instruction execution. A trace exception occurs if instruction tracing is enabled ($T0 = 0$, $T1 = 1$) when the STOP instruction begins execution. If an interrupt request is asserted with a priority higher than the priority level set by the new status register value, an interrupt exception occurs; otherwise, the interrupt request is ignored. External reset always initiates reset exception processing.

Condition Codes:
Set according to the immediate operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
IMMEDIATE DATA															

Instruction Fields:
Immediate field — Specifies the data to be loaded into the status register.

SECTION 5

CPU32 INSTRUCTIONS

This section describes the instructions provided for the CPU32. The CPU32 can execute object code from an MC68000 and MC68010 and many of the instructions of the MC68020.

Three new instructions are provided for use with the CPU32. These are the enter background mode (BGND), low-power stop (LPSTOP), and table lookup and interpolate (TBLS, TBLSN, TBLU, and TBLUN). Table 5-1 lists the MC68020 instructions not supported by the CPU32.

Table 5-1. MC68020 Instructions Not Supported

Opcode	Description
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
CALLM	CALL Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
cpBcc	Branch on Coprocessor Condition
cpDBcc	Test Coprocessor Condition Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRAPcc	Trap on Coprocessor Condition
RTM	Return from Module
PACK	Pack BCD
UNPK	Unpack BCD

Addressing in the CPU32 is register oriented. Most instructions allow the results of the specified operation to be placed either in a register or directly in memory. This flexibility eliminates the need for extra instructions to store register contents in memory. Table 5-2 lists the M68000 addressing modes with cross-references to the MC68000, MC68010, CPU32, and MC68020. When referring to instructions in the previous sections, refer to Table 5-2 to identify the addressing modes available to the CPU32.

Table 5-3 lists the instructions for the CPU32.

Table 5-2. M68000 Addressing Modes

Addressing Mode	Syntax	MC68000 MC68010	CPU32	MC68020
Register Indirect	Rn	X	X	X
Address Register Indirect	(An)	X	X	X
Address Register Indirect with Postincrement	(An) +	X	X	X
Address Register Indirect with Postdecrement	– (An)	X	X	X
Address Register Indirect with Displacement	(d ₁₆ ,An)	X	X	X
Address Register Indirect with Index (8-Bit Displacement)	(dg,An,Xn)	X	X	X
Address Register Indirect with Index (Base Displacement)	(dg,An,Xn*SCALE)		X	X
Memory Indirect with Postincrement	([bd,An],Xn,od)			X
Memory Indirect with Preincrement	([bd,An],Xn,od)			X
Absolute Short	(xxx).W	X	X	X
Absolute Long	(xxx).L	X	X	X
Program Counter Indirect with Displacement	(d ₁₆ ,PC)	X	X	X
Program Counter Indirect with Index (8-Bit Displacement)	(dg,PC,Xn)	X	X	X
Program Counter Indirect with Index (Base Displacement)	(dg,PC,Xn*SCALE)		X	X
Immediate	#(data)	X	X	X
PC Memory Indirect with Postincrement	([bd,PC],Xn,od)			X
PC Memory Indirect with Predecrement	([bd,PC],Xn,od)			X

NOTE: Xn,SIZE*SCALE — Denotes index register n (data or address), the index size (W for word, L for long word) and scale factor (1, 2, 4, or 8 for no word, long-word, or 8 for quad-word scaling, respectively).

X = Supported

Table 5-3. CPU32 Instruction Set

Mnemonic	Description	Mnemonic	Description
ABCD	Add Decimal with Extend	MOVE	Move
ADD	Add	MOVEA	Move Address
ADDA	Add Address	MOVE from CCR	Move Condition Code Register
ADDI	Add Immediate	MOVE from SR	Move from Status Register
ADDQ	Add Quick	MOVE to SR	Move to Status Register
ADDX	Add with Extend	MOVE USP	Move User Stack Pointer
AND	Logical AND	MOVEC	Move Control Register
ANDI	Logical AND Immediate	MOVEM	Move Multiple Registers
ANDI to CCR	AND Immediate to Condition Code	MOVEP	Move Peripheral
ANDI to SR	AND Immediate to Status Register	MOVEQ	Move Quick
ASL, ASR	Arithmetic Shift Left and Right	MOVES	Move Alternate Address Space
Bcc	Branch Conditionally	MULS	Signed Multiply
BCHG	Test Bit and Change	MULU	Unsigned Multiply
BCLR	Test Bit and Clear	NBCD	Negate Decimal with Extend
BGND	Enter Background Mode	NEG	Negate
BKPT	Breakpoint	NEGX	Negate with Extend
BRA	Branch	NOP	No Operation
BSET	Test Bit and Set	NOT	Logical Complement
BSR	Branch to Subroutine	PEA	Push Effective Address
BTST	Test Bit	RESET	Reset External Devices
CHK	Check Register Against Bound	ROL, ROR	Rotate Left and Right
CHK2	Check Register Against Upper and Lower Bounds	ROXL, ROXR	Rotate with Extend Left and Right
CLR	Clear	RTD	Return and Deallocate
CMP	Compare	RTE	Return from Exception
CMPA	Compare Address	RTR	Return and Restore Codes
CMPI	Compare Immediate	RTS	Return from Subroutine
CMPPM	Compare Memory to Memory	SBCD	Subtract Decimal with Extend
CMP2	Compare Register Against Upper and Lower Bounds	Scc	Set Conditionally
DBcc	Test Condition, Decrement and Branch	STOP	Stop
DIVS, DIVSL	Signed Divide	SUB	Subtract
DIVU, DIVUL	Unsigned Divide	SUBA	Subtract Address
EOR	Logic Exclusive OR	SUBI	Subtract Immediate
EORI	Logical Exclusive OR Immediate	SUBQ	Subtract Quick
EORI to CCR	Exclusive OR Immediate to Condition Code	SUBX	Subtract with Extend
EORI to SR	Exclusive OR Immediate to Status Register	SWAP	Swap Register Words
EXG	Exchange Registers	TAS	Test Operand and Set
EXT, EXTB	Sign Extend	TBLS, TBLSN	Signed/Unsigned Table Lookup and Interpolate
ILLEGAL	Table Illegal Instruction Trap	TBLU, TBLUN	Signed/Unsigned Table Lookup and Interpolate
JMP	Jump	TRAP	Trap
JSR	Jump to Subroutine	TRAPcc	Trap Conditionally
LEA	Load Effective Address	TRAPV	Trap on Overflow
LINK	Link and Allocate	TST	Test Operand
LPSTOP	Low Power Stop	UNLK	Unlink
LSL, LSR	Logical Shift Left and Right		

Operation: if (background mode enabled) then
 Enter Background Mode

 else
 Format/Vector offset \blacktriangleright – (SSP)
 PC \blacktriangleright – (SSP)
 SR \blacktriangleright – (SSP)
 (Vector) \blacktriangleright PC

Assembler
Syntax: BGND

Atributes: Size = (Unsize)

Description: The processor suspends instruction execution and enters back-ground mode (if enabled). The freeze output is asserted to acknowledge entrance into background mode. Upon exiting background mode, instruction execution continues with the instruction pointed to by the current program counter.

If background mode is not enabled, the processor initiates illegal instruction exception processing. The vector number is generated to reference the illegal instruction exception vector. Refer to appropriate user’s manual for detailed information on background mode.

5

Condition Codes:

X	N	Z	V	C
—	—	—	—	—

- X Not affected
- N Not affected
- Z Not affected
- V Not affected
- C Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	0	1	0

LPSTOP

Low-Power Stop
(CPU32)

LPSTOP

Operation: if supervisor state
Immediate Data \blacktriangleright SR
Interrupt Mask \blacktriangleright External Bus Interface (EBI)
STOP
else TRAP

Assembler
Syntax: LPSTOP #<data>

Attributes: Size = (Word) Privileged

Description: The immediate operand is moved into the entire status register, the program counter is advanced to point to the next instruction, and the processor stops fetching and executing instructions. A CPU LPSTOP broadcast cycle is executed to CPU space \$3 to copy the updated interrupt mask to the external bus interface (EBI). The internal clocks are stopped.

Execution of instructions resumes when a trace, interrupt, or reset exception occurs. A trace exception will occur if the trace state is on when the LPSTOP instruction is executed. If an interrupt request is asserted with a higher priority than the current priority level set by the new status register value, an interrupt exception occurs; otherwise the interrupt request is ignored. If the bit of the immediate data corresponding to the S bit is off, execution of the instruction will cause a privilege violation. An external reset always initiates reset exception processing.

Condition Codes:
Set according to the immediate operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
IMMEDIATE DATA															

Instruction Fields:
Immediate field:
Specifies the data to be loaded into the status register.

Operation: Rounded:
 $\text{ENTRY}(n) + \{(\text{ENTRY}(n+1) - \text{ENTRY}(n)) \cdot \text{Dx}[7:0]\} / 256 \nabla \text{Dx}$
 Unrounded:
 $\text{ENTRY}(n) \cdot 256 + \{(\text{ENTRY}(n+1) - \text{ENTRY}(n)) \cdot \text{Dx}[7:0]\} \nabla \text{Dx}$

Where $\text{ENTRY}(n)$ and $\text{ENTRY}(n+1)$ are either:

1. Consecutive entries in the table pointed to by the $\langle \text{ea} \rangle$ and indexed by $\text{Dx}[15:8] \cdot \text{size}$ or,
2. The registers Dym, Dyn respectively

Assembler	TBLS.<size>	<ea>,Dx	*Result rounded
Syntax:	TBLSN.<size>	<ea>,Dx	*Result not rounded
	TBLS.<size>	Dym:Dyn, Dx	*Result rounded
	TBLSN.<size>	Dym:Dyn, Dx	*Result not rounded

Attributes: Size = (Byte, Word, Long)

Description: The signed table lookup and interpolate instruction, TBLS, allows the efficient use of piecewise linear, compressed data tables to model complex functions. The TBLS instruction has two modes of operation: table lookup and interpolate mode and data register interpolate mode.

5

For table lookup and interpolate mode, data register $\text{Dx}[15:0]$ contains the independent variable X . The effective address points to the start of a signed byte, word, or long-word table containing a linearized representation of the dependent variable, Y , as a function of X . In general, the independent variable, located in the low-order word of Dx , consists of an 8-bit integer part and an 8-bit fractional part. An assumed radix point is located between bits 7 and 8. The integer part, $\text{Dx}[15:8]$, is scaled by the operand size and is used as an offset into the table. The selected entry in the table is subtracted from the next consecutive entry. A fractional portion of this difference is taken by multiplying by the interpolation fraction, $\text{Dx}[7:0]$. The adjusted difference is then added to the selected table entry. The result is returned in the destination data register, Dx .

For register interpolate mode, the interpolation occurs using the Dym and Dyn registers in place of the two table entries. For this mode, only the fractional portion, Dx[7:0], is used in the interpolation, and the integer portion, Dx[15:8], is ignored. The register interpolation mode may be used with several table lookup and interpolations to model multidimensional functions.

Signed table entries range from -2^{n-1} to $2^{n-1}-1$; whereas, unsigned table entries range from 0 to 2^n-1 where n is 8, 16, or 32 for byte, word, and long-word tables, respectively.

Rounding of the result is optionally selected via the “R” instruction field. If R=0 (TABLE), the fractional portion is rounded according to the round-to-nearest algorithm. The rounding procedure can be summarized by the following table.

Adjusted Difference Fraction	Rounding Adjustment
$\leq -1/2$	-1
$> -1/2$ and $< 1/2$	+0
$\geq 1/2$	+1

The adjusted difference is then added to the selected table entry. The rounded result is returned in the destination data register, Dx. Only the portion of the register corresponding to the selected size is affected.

	31	24 23	16 15	8 7	0
BYTE	UNAFFECTED	UNAFFECTED	UNAFFECTED	RESULT	
WORD	UNAFFECTED	UNAFFECTED	RESULT	RESULT	
LONG	RESULT	RESULT	RESULT	RESULT	

If R = 1 (TABLENR), the result is returned in register Dx without rounding. If the size is byte, the integer portion of the result is returned in Dx(15:8); the integer portion of a word result is stored in Dx(23:8); the least significant 24 bits of a long result are stored in Dx(31:8). Byte and word results are sign extended to fill the entire 32-bit register.

	31	24 23	16 15	8 7	0
BYTE	SIGN EXTENDED		SIGN EXTENDED		FRACTION
WORD	SIGN EXTENDED		RESULT		FRACTION
LONG	RESULT		RESULT		FRACTION

NOTE

The long-word result contains only the least significant 24 bits of integer precision.

For all sizes, the 8-bit fractional portion of the result is returned in the low byte of the data register, Dx(7:0). User software can make use of the fractional data to reduce cumulative errors in lengthy calculations or implement rounding algorithms different from that provided by other forms of TBLS. The assumed radix point described previously places two restrictions on the programmer:

- 1) Tables are limited to 257 entries in length.
- 2) Interpolation resolution is limited to 1/256 the distance between consecutive table entries. The assumed radix point should not, however, be construed by the programmer as a requirement that the independent variable be calculated as a fractional number in the range $0 \leq X \leq 255$. On the contrary, X should be considered to be an integer in the range $0 \leq X \leq 65535$; realizing that the table is actually a compressed representation of a linearized function in which only every 256th value is actually stored in memory.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

- X Not affected.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if the integer portion of an unrounded long result is not in the range, $-(2^{23}) \leq \text{Result} \leq (2^{23}) - 1$. Cleared otherwise.
- C Always cleared.

Instruction Format:

Table Lookup and Interpolate:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	REGISTER Dx			1	R	0	1	SIZE		0	0	0	0	0	0

Data Register Interpolate:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	REGISTER Dym		
0	REGISTER Dx			1	R	0	0	SIZE		0	0	0	REGISTER Dyn		

Instruction Fields:

Effective address field (table lookup and interpolate mode only):
Specifies the destination location. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An*	—	—	(xxx).L	111	001
(An)	—	—	#(data)	—	—
(An) +	—	—			
– (An)	100	reg. number:An			
(d16,An)	101	reg. number:An	(d16,PC)	111	010
(dg,An,Xn)	110	reg. number:An	(dg,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011

Size field:
Specifies the size of operation.
00 — Byte Operation
01 — Word Operation
10 — Long Operation

Register field:
Specifies the destination data register, Dx. On entry, the register contains the interpolation fraction and entry number.

Dym, Dyn field:
If the effective address mode field is nonzero, this operand register is unused and should be zero. If the effective address mode field is zero, the surface interpolation variant of this instruction is implied, and Dyn specifies one of the two source operands.

Rounding mode field:
The 'R' bit controls the rounding of the final result. When R = 0, the result is rounded according to the round-to-nearest algorithm. When R = 1, the result is returned unrounded.

Operation: Rounded:
 $\text{ENTRY}(n) + \{(\text{ENTRY}(n+1) - \text{ENTRY}(n)) \cdot \text{Dx}[7:0]\} / 256 \blacklozenge \text{Dx}$
 Unrounded:
 $\text{ENTRY}(n) \cdot 256 + \{(\text{ENTRY}(n+1) - \text{ENTRY}(n)) \cdot \text{Dx}[7:0]\} \blacklozenge \text{Dx}$

Where ENTRY(n) and ENTRY(n+1) are either:

1. Consecutive entries in the table pointed to by the <ea> and indexed by Dx[15:8]*size or,
2. The registers Dym, Dyn respectively

Assembler	TBLU.<size>	<ea>,Dx	* Result rounded
Syntax:	TBLUN.<size>	<ea>,Dx	* Result not rounded
	TBLU.<size>	Dym:Dyn, Dx	* Result rounded
	TBLUN.<size>	Dym:Dyn, Dx	* Result not rounded

Attributes: Size = (Byte, Word, Long)

Description: The unsigned table lookup and interpolate instruction, TBLU, allows the efficient use of piecewise linear, compressed data tables to model complex functions. The TBLU instruction has two modes of operation: table lookup and interpolate mode and data register interpolate mode.

For table lookup and interpolate mode, data register Dx[15:0] contains the independent variable X. The effective address points to the start of a unsigned byte, word, or long-word table containing a linearized representation of the dependent variable, Y, as a function of X. In general, the independent variable, located in the low-order word of Dx, consists of an 8-bit integer part and an 8-bit fractional part. An assumed radix point is located between bits 7 and 8. The integer part, Dx[15:8], is scaled by the operand size and is used as an offset into the table. The selected entry in the table is subtracted from the next consecutive entry. A fractional portion of this difference is taken by multiplying by the interpolation fraction, Dx[7:0]. The adjusted difference is then added to the selected table entry. The result is returned in the destination data register, Dx.

For register interpolate mode, the interpolation occurs using the Dym and Dyn registers in place of the two table entries. For this mode, only the fractional portion, Dx[7:0], is used in the interpolation, and the integer portion, Dx[15:8], is ignored. The register interpolation mode may be used with several table lookup and interpolations to model multidimensional functions.

Signed table entries range from -2^{n-1} to $2^{n-1} - 1$; whereas, unsigned table entries range from 0 to $2^n - 1$ where n is 8, 16, or 32 for byte, word, and long-word tables, respectively. The unsigned and unrounded table results will be zero extended instead of sign extended.

Rounding of the result is optionally selected via the "R" instruction field. If $R=0$ (TABLE), the fractional portion is rounded according to the round-to-nearest algorithm. The rounding procedure can be summarized by the following table.

Adjusted Difference Fraction	Rounding Adjustment
$\geq 1/2$	+ 1
$< 1/2$	+ 0

The adjusted difference is then added to the selected table entry. The rounded result is returned in the destination data register, Dx. Only the portion of the register corresponding to the selected size is affected.

	31	24	23	16	15	8	7	0
BYTE	UNAFFECTED		UNAFFECTED		UNAFFECTED		RESULT	
WORD	UNAFFECTED		UNAFFECTED		RESULT		RESULT	
LONG	RESULT		RESULT		RESULT		RESULT	

If $R=1$ (TBLUN), the result is returned in register Dx without rounding. If the size is byte, the integer portion of the result is returned in Dx(15:8); the integer portion of a word result is stored in Dx(23:8); the least significant 24 bits of a long result are stored in Dx(31:8). Byte and word results are sign extended to fill the entire 32-bit register.

	31	24	23	16	15	8	7	0
BYTE	SIGN EXTENDED		SIGN EXTENDED		RESULT		FRACTION	
WORD	SIGN EXTENDED		RESULT		RESULT		FRACTION	
LONG	RESULT		RESULT		RESULT		FRACTION	

NOTE

The long-word result contains only the least significant 24 bits of integer precision.

For all sizes, the 8-bit fractional portion of the result is returned in the low byte of the data register, Dx(7:0). User software can make use of the fractional data to reduce cumulative errors in lengthy calculations or implement rounding algorithms different from that provided by other forms of TBLS. The assumed radix point described previously places two restrictions on the programmer:

- 1) Tables are limited to 257 entries in length.
- 2) Interpolation resolution is limited to 1/256 the distance between consecutive table entries. The assumed radix point should not, however, be construed by the programmer as a requirement that the independent variable be calculated as a fractional number in the range $0 \leq X \leq 255$. On the contrary, X should be considered to be an integer in the range $0 \leq X \leq 65535$; realizing that the table is actually a compressed representation of a linearized function in which only every 256th value is actually stored in memory.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

- X Not affected.
- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if the integer portion of an unrounded long result is not in the range, $-(2^{23}) \leq \text{Result} \leq (2^{23}) - 1$. Cleared otherwise.
- C Always cleared.

Instruction Format:

Table Lookup and Interpolate:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	REGISTER Dx			0	R	0	1	SIZE		0	0	0	0	0	0

Data Register Interpolate:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	REGISTER Dym		
0	REGISTER Dx			0	R	0	0	SIZE		0	0	0	REGISTER Dyn		

Instruction Fields:

Effective address field (table lookup and interpolate mode only):

Specifies the destination location. Only control addressing modes are allowed as shown:

Addressing Mode	Mode	Register
Dn	—	—
An*	—	—
(An)	010	reg. number:An
(An) +	—	—
– (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(dg,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#(data)	—	—
(d ₁₆ ,PC)	111	010
(dg,PC,Xn)	111	011
(bd,PC,Xn)	111	011

Size field:

Specifies the size of operation.

00 — Byte Operation

01 — Word Operation

10 — Long Operation

Register field:

Specifies the destination data register, Dx. On entry, the register contains the interpolation fraction and entry number.

Dym, Dyn field:

If the effective address mode field is nonzero, this operand register is unused and should be zero. If the effective address mode field is zero, the surface interpolation variant of this instruction is implied, and Dyn specifies one of the two source operands.

Rounding mode field:

The 'R' bit controls the rounding of the final result. When R = 0, the result is rounded according to the round-to-nearest algorithm. When R = 1, the result is returned unrounded.

SECTION 6

INSTRUCTION FORMAT SUMMARY

This section contains a listing of the M68000 instructions in binary format listed in alphabetical order and operation code map for the M68000 instruction set.

6.1 INSTRUCTION FORMAT

The following paragraphs present a summary of the binary encoding fields.

6.1.1 Coprocessor ID Field

This field specifies which coprocessor in a system is to perform the operation. When using directly supported floating-point instructions for the MC68040, this field must be set to one.

6.1.2 Effective Address Field

This field specifies which addressing mode is to be used. For some operations, restrictions are placed on which of the available addressing modes are allowed. These restrictions are enforced by hardware.

6.1.3 Register/Memory Field

This field is common to all of the arithmetic instructions. A zero in this field indicates that the operation is register to register and a one indicates that the operations is (ea) to register.

6

6.1.4 Source Specifier Field

This field is common to all of the arithmetic instructions. The definition of this field is affected by the value of the register/memory (R/M) field. If

R/M = 0, specifies the source floating-point data register (FPDR). If R/M = 1, specifies the source operand data format:

000	L	Long-Word Integer
001	S	Single-Precision Real
010	X	Extended-Precision Real
011	P	Packed-Decimal Real
100	W	Word Integer
101	D	Double-Precision Real
110	B	Byte Integer

6.1.5 Destination Register Field

This field is common to all of the arithmetic instructions. This field specifies the FPDR that is to be used as the destination. The results is always stored in this register.

6.1.6 Conditional Predicate Field

This field is common to all of the conditional instructions and specifies the conditional test that is to be evaluated. Table 6-1 shows the binary encodings for the conditional tests.

6.1.7 Shift and Rotate Instructions

The following paragraphs define the fields used with the shift and rotate instructions.

6.1.7.1 COUNT REGISTER FIELD. If $i/r = 0$, this field contains the rotate (shift) count of 1–8 (a zero specifies 8). If $i/r = 1$, this field specifies a data register that contains the rotate (shift) count. The following shift and rotate fields are encoded as:

- dr field 0—Rotate (shift) Right
- 1—Rotate (shift) Left
- i/r field 0—Immediate Rotate (shift) Count
- 1—Register Rotate (shift) Count

6.1.7.2 REGISTER FIELD. Specifies a data register to be rotated (shifted).

Table 6-1. Conditional Predicate Field Encoding

Conditional Predicate	Mnemonic	Definition
000000	F	False
000001	EQ	Equal
000010	OGT	Ordered Greater Than
000011	OGE	Ordered Greater Than or Equal
000100	OLT	Ordered Less Than
000101	OLE	Ordered Less Than or Equal
000110	OGL	Ordered Greater Than or Less Than
000111	OR	Ordered
001000	UN	Unordered
001001	UEQ	Unordered or Equal
001010	UGT	Unordered or Greater Than
001011	UGE	Unordered or Greater Than or Equal
001100	ULT	Unordered or Less Than
001101	ULE	Unordered or Less Than or Equal
001110	NE	Not Equal
001111	T	True
010000	SF	Signaling False
010001	SEQ	Signaling Equal
010010	GT	Greater Than
010011	GE	Greater Than or Equal
010100	LT	Less Than
010101	LE	Less Than or Equal
010110	GL	Greater Than or Less Than
010111	GLE	Greater Than or Less Than or Equal
011000	NGLE	Not (Greater Than or Less Than or Equal)
011001	NGL	Not (Greater Than or Less Than)
011010	NLE	Not (Less Than or Equal)
011011	NLT	Not (Less Than)
011100	NGE	Not (Greater Than or Equal)
011101	NGT	Not (Greater Than)
011110	SNE	Signaling Not Equal
011111	ST	Signaling True

6.1.8 Size Field

This field specifies the size of the operation. The encoding is as follows:

- 00—Byte Operation
- 01—Word Operation
- 10—Long Operation

6.1.9 Opmode Field

Refer to the applicable instruction in the previous sections for the encoding of this field.

6.1.10 Address/Data (A/D) Field

This field specifies the type of general register. The encoding is as follows:

- 0—Data Register
- 1—Address Register

6.2 OPERATION CODE MAP

Table 6-2 list the encoding for bits 15–12 and the operation performed.

Table 6-2. Operation Code Map

Bits 15 through 12	Operation
0000	Bit Manipulation/MOVP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/Scc/DBcc/TRAPcc
0110	Bcc/BSR/BRA
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned, Reserved)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate/Bit Field
1111	Coprocessor Interface

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER Rx			1	0	0	0	0	R/M	REGISTER Ry		

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE		REGISTER			

ADDA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE			REGISTER		

ADDI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS					
									MODE			REGISTER			
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	DATA			0	SIZE			EFFECTIVE ADDRESS				
										MODE			REGISTER		

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	REGISTER Rx			1	SIZE		0	0	R/M	REGISTER Ry		

Shift/Rotate Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/REGISTER			dr	SIZE		I/R	TYPE		REGISTER Ry		

Shift/Rotate Memory

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	TYPE		dr	1	1	EFFECTIVE ADDRESS MODE				REGISTER	

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER			OPMODE			EFFECTIVE ADDRESS MODE				REGISTER	

ANDI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	SIZE		EFFECTIVE ADDRESS					
									MODE		REGISTER				
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

ANDI to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	BYTE DATA (8 BITS)							

ANDI to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
WORD DATA (16 BITS)															

ASL, ASR

Register Shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/REGISTER			dr	SIZE		i/r	0	0	REGISTER		

Memory Shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	EFFECTIVE ADDRESS			REGISTER		
										MODE					

Bcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	CONDITION				8-BIT DISPLACEMENT							
16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00															
32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF															

BCHG

Bit Number Dynamic, specified in a register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	0	1	EFFECTIVE ADDRESS MODE			REGISTER		

Bit Number Static, specified as immediate data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	EFFECTIVE ADDRESS MODE			REGISTER		
0	0	0	0	0	0	0	0	BIT NUMBER							

BCLR

Bit Number Dynamic, specified in a register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	1	0	EFFECTIVE ADDRESS MODE			REGISTER		

Bit Number Static, specified as immediate data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	0	0	0	0	0	0	BIT NUMBER							

BCLR

Bit Number Dynamic, specified in a register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	1	0	EFFECTIVE ADDRESS MODE		REGISTER			

Bit Number Static, specified as immediate data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	0	BIT NUMBER							

BFCHG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	Do	OFFSET					Dw	WIDTH				

BFCLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	0	0	Do	OFFSET					Dw	WIDTH				

BFEXTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	1	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	REGISTER			Do	OFFSET					Dw	WIDTH				

BFEXTU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	1	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	REGISTER			Do	OFFSET					Dw	WIDTH				

BFFFO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	REGISTER			Do	OFFSET					Dw	WIDTH				

BFINS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	REGISTER			Do	OFFSET					Dw	WIDTH				

BFSET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	Do	OFFSET					Dw	WIDTH				

BFTST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	0	1	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	Do	OFFSET					Dw	WIDTH				

BGND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	0	1	0

BKPT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	VECTOR		

BRA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8-BIT DISPLACEMENT							
16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00															
32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF															

BSET

Bit Number Dynamic, specified in a register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Bit Number Static, specified as immediate data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	BIT NUMBER								

BSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8-BIT DISPLACEMENT							
16-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$00															
32-BIT DISPLACEMENT IF 8-BIT DISPLACEMENT = \$FF															

BTST

Bit Number Dynamic, specified in a register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REGISTER			1	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Bit Number Static, specified as immediate data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	0	0	0	0	0	0	BIT NUMBER							

CAS,CAS2

CAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	SIZE		0	1	1	EFFECTIVE ADDRESS MODE				REGISTER	
0	0	0	0	0	0	0	Du			0	0	0	Dc		

CAS2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	SIZE		0	1	1	1	1	1	1	0	0
D/A1	Rn1			0	0	0	Du1			0	0	0	Dc1		
D/A2	Rn2			0	0	0	Du2			0	0	0	Dc2		

CHK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	REGISTER			SIZE		0	EFFECTIVE ADDRESS MODE				REGISTER	

CHK2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS MODE				REGISTER	
D/A	REGISTER			1	0	0	0	0	0	0	0	0	0	0	0

CINV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	CACHE		0	SCOPE		REGISTER		

CLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	SIZE		EFFECTIVE ADDRESS MODE				REGISTER	

CMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS						
											MODE		REGISTER			

CMPA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
											MODE		REGISTER		

CMPI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

CMPM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER Ax			1	SIZE		0	0	1	REGISTER Ay		

CMP2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		
D/A	REGISTER			0	0	0	0	0	0	0	0	0	0	0	0

6

cpBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	1	SIZE	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
WORD OR															
LONG WORD DISPLACEMENT															

CMP2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	SIZE		0	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		
D/A	REGISTER			0	0	0	0	0	0	0	0	0	0		

cpBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	1	SIZE	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
WORD OR															
LONG WORD DISPLACEMENT															

cpDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID =			0	0	1	0	0	1	REGISTER		
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
DISPLACEMENT (16 BIT)															

cpGEN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
COPROCESSOR-DEPENDENT COMMAND WORD															
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSION WORDS															

cpRESTORE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID			1	0	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

cpSAVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠ 000			1	0	0	EFFECTIVE ADDRESS MODE REGISTER					

cpScc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠ 000			0	0	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION					
OPTIONAL EFFECTIVE ADDRESS OR COPROCESSOR-DEFINED EXTENSION WORDS															

cpTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠ 000			0	0	1	1	1	1	OPMODE		
0	0	0	0	0	0	0	0	0	0	COPROCESSOR CONDITION					
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS															
OPTIONAL WORD															
OR LONG WORD OPERAND															

CPUSH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	CACHE		1	SCOPE		REGISTER		

DBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	CONDITION				1	1	0	0	1	REGISTER		
DISPLACEMENT (16 BITS)															

DIVS, DIVSL

Word Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

Long Word Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	EFFECTIVE ADDRESS MODE			REGISTER		
0	REGISTER Dq			1	SIZE	0	0	0	0	0	0	0	REGISTER Dr		

DIVU, DIVUL

Word Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER			0	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

Long Word Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	EFFECTIVE ADDRESS MODE			REGISTER		
0	REGISTER Dq			0	SIZE	0	0	0	0	0	0	0	REGISTER Dr		

EOR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	REGISTER			OPMODE			EFFECTIVE ADDRESS MODE			REGISTER		

EORI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	SIZE		EFFECTIVE ADDRESS					
									MODE		REGISTER				
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

EORI to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	BYTE DATA (8 BITS)							

EORI to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
WORD DATA (16 BITS)															

EXG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER Rx			1	OPMODE					REGISTER Ry		

EXT, EXTB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	OPMODE			0	0	0	REGISTER		

FABS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FACOS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	0	0

FADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
		MODE		REGISTER											
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FASIN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	1	1	0	0	

FATAN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	0	1	0

FATANH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	0	1

FBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	1	SIZE	CONDITIONAL PREDICATE						
16-BIT DISPLACEMENT, OR MOST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT															
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)															

FCMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	1	1	0	0	0

FCOS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	0	1

FCOSH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	0	0	1

FDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	1	0	0	1	COUNT REGISTER		
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICATE					
16-BIT DISPLACEMENT															

6

FDIV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FETOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	0	0

FETOXM1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	0	0

FGETEXP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
									MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	1	0

FGETMAN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
				MODE			REGISTER								
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	1	1	1	1

FINT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
				MODE			REGISTER								
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	0	1

FINTRZ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	0	1	1

FLOG10

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	1	0	1

FLOG2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	1	1	0

FLOGN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	1	0	0

FLOGNP1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	0	1	1	0

FMOD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
				MODE			REGISTER								
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	0	0	1

FMOVE (Data Register)

Effective Address to Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE REGISTER						
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

Register to Memory

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
				MODE		REGISTER									
0	1	1	DESTINATION FORMAT		SOURCE REGISTER		K FACTOR (IF REQUIRED)								

FMOVE (System Control Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS						
				MODE		REGISTER									
1	0	dr	REGISTER SELECT		0	0	0	0	0	0	0	0	0	0	0

FMOVECR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	0	0	0	0	0	0	0
				MODE		REGISTER									
0	1	0	1	1	1	DESTINATION REGISTER		ROM OFFSET							

FMOVEM (Data Registers)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODEREGISTER					
1	1	dr	MODE		0	0	0	REGISTER LIST							

FMOVEM (Control Registers)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS					
										MODE	REGISTER				
1	0	dr	REGISTER LIST			0	0	0	0	0	0	0	0	0	0

FMUL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FNEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FNOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FREM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	0	1

FRESTORE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			1	0	1	EFFECTIVE ADDRESS MODE			REGISTER		

FSAVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		

FSCALE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	1	0

FScC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	1	EFFECTIVE ADDRESS MODE			REGISTER		
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICATE					

FSGLDIV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	0	0	1	0	0

FSGLMUL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	1	0	0	1	1	1	

FSIN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	1	1	1	1	0

FSINCOS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER, FP _s			0	1	1	0	DESTINATION REGISTER, FP _c			

FSINH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID		0	0	0	EFFECTIVE ADDRESS MODE		REGISTER				
0	R/M	0	SOURCE SPECIFIER		DESTINATION REGISTER			0	0	0	0	0	1	0	

FSQRT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FSUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			OPMODE						

FTAN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	1	1	1

FTANH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	0	1	0	0	1

FTENTOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	1	0

FTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	1	1	1	1	MODE		
0	0	0	0	0	0	0	0	0	0	CONDITIONAL PREDICATE					
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)															
LEAST SIGNIFICANT WORD OR 32-BIT OPERAND (IF NEEDED)															

FTST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	1	1	1	0	1	0

FTWOTOX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	COPROCESSOR ID			0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	R/M	0	SOURCE SPECIFIER			DESTINATION REGISTER			0	0	1	0	0	0	1

ILLEGAL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

JMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

JSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	EFFECTIVE ADDRESS MODE			REGISTER		

LEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

LINK

Word Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	REGISTER		
WORD DISPLACEMENT															

Long Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	1	REGISTER		
HIGH-ORDER DISPLACEMENT															
LOW-ORDER DISPLACEMENT															

LPSTOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
IMMEDIATE DATA															

LSL, LSR

Register Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/REGISTER			dr	SIZE			i/r	0	1	REGISTER	

Memory Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	dr	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

MOVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE			DESTINATION					SOURCE					
				REGISTER			MODE			MODE			REGISTER		

MOVEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE			DESTINATION REGISTER			0	0	1	SOURCE MODE			REGISTER	

MOVE from CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

MOVE to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

MOVE from SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	EFFECTIVE ADDRESS					
										MODE			REGISTER		

MOVE to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

MOVE16

Postincrement Source and Destination

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	0	0	1	0	0	REGISTER Ax		
1	REGISTER Ay			0	0	0	0	0	0	0	0	0	0	0	0

Absolute Long Address Source or Destination

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	0	0	0	OPMODE		REGISTER Ay		
HIGH-ORDER ADDRESS															
LOW-ORDER ADDRESS															

MOVE USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	REGISTER		

MOVEC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D		REGISTER			CONTROL REGISTER										

MOVEM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	SIZE	EFFECTIVE ADDRESS					
										MODE		REGISTER			
REGISTER LIST MASK															

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	DATA REGISTER			OPMODE			0	0	1	ADDRESS REGISTER		
DISPLACEMENT (16 BITS)															

MOVEQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	REGISTER			0	DATA							

MOVES

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE		REGISTER			
A/D	REGISTER			dr	0	0	0	0	0	0	0	0	0	0	0

MULS

Word Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER			1	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

Long Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	REGISTER DI			1	SIZE	0	0	0	0	0	0	0	REGISTER Dh		

MULU

Word Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	REGISTER			0	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

Long Form

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		
0	REGISTER DI			0	SIZE	0	0	0	0	0	0	0	REGISTER Dh		

NBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	EFFECTIVE ADDRESS MODE			REGISTER		

6

NEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	SIZE		EFFECTIVE ADDRESS MODE			REGISTER		

NEGX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		

NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE		REGISTER			

OR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER			OPMODE			EFFECTIVE ADDRESS					
										MODE		REGISTER			

ORI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	SIZE		EFFECTIVE ADDRESS MODE REGISTER					
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

ORI to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	BYTE DATA (8 BITS)							

ORI to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
WORD DATA (16 BITS)															

PACK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER Dy/Ay		1	0	1	0	0	R/M	REGISTER Dx/Ax			
16-BIT EXTENSION: ADJUSTMENT															

PBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	1	SIZE	MC68851 CONDITION					
16-BIT DISPLACEMENT, OR MOST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT															
LEAST SIGNIFICANT WORD OF 32-BIT DISPLACEMENT (IF NEEDED)															

PDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	0	0	1	COUNT REGISTER		
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION					
16-BIT DISPLACEMENT															

PEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	EFFECTIVE ADDRESS MODE				REGISTER	

PFLUSH (MC68030)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	1	MODE			0	0	MASK			FC				

PFLUSH (MC68040)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	0	0	0	OPMODE		REGISTER		

**PFLUSH
PFLUSHA
PFLUSHS (MC68851)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	0	1	MODE			0	MASK			FC					

PFLUSHR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

PLOAD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
											MODE		REGISTER		
0	0	1	0	0	0	R/W	0	0	0	0	FC				

PMOVE (MC68030)

For SRP, CRP, and TC Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	0	P REG			R/W	FD	0	0	0	0	0	0	0	0

For MMUSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	0	0

PMOVE (MC68030)

For TT Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	0	0	P REG			R/W	FD	0	0	0	0	0	0	0	0

PMOVE (MC68851)

For TC, CRP, DRP, SRP, CAL, VAL, Scc, and AC Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	0	P REG			R/W	0	0	0	0	0	0	0	0	0

For BADx and BACx Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	1	P REG			R/W	0	0	0	0	NUM			0	0

For PSR and PCSR Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	1	1	P REG			R/W	0	0	0	0	0	0	0	0	0

PRESTORE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	1	0	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

PSAVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	1	0	0	EFFECTIVE ADDRESS					
										MODE		REGISTER			

PScC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	EFFECTIVE ADDRESS MODE REGISTER					
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION					

PTEST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
1	0	0	LEVEL			R/W	A	REG			FC				

PTEST (MC68040)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	0	1	R/W	0	1	REGISTER		

PTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	1	1	1	OPMODE		
0	0	0	0	0	0	0	0	0	0	MC68851 CONDITION					
16-BIT OPERAND OR MOST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)															
LEAST SIGNIFICANT WORD OF 32-BIT OPERAND (IF NEEDED)															

PVALID

VAL Contains Access Level to Test Against

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Main Processor Register Contains Access Level to Test Against

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	0	1	0	1	0	0	0	0	0	0	0	0	REG		

RESET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

ROL, ROR

Register Rotate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/ REGISTER			dr	SIZE		i/r	1	1	REGISTER		

Memory Rotate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	dr	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

ROXL, ROXR

Register Rotate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	COUNT/ REGISTER			dr	SIZE		i/r	1	0	REGISTER		

Memory Rotate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	dr	1	1	EFFECTIVE ADDRESS MODE			REGISTER		

RTD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0
DISPLACEMENT (16 BITS)															

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

Format/Offset Word (in stack frame)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FORMAT				0	0	VECTOR OFFSET									

RTM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	D/A	REGISTER		

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER Dy/Ay			1	0	0	0	0	R/M	REGISTER Dx/Ax		

Scc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	CONDITION				1	1	EFFECTIVE ADDRESS MODE				REGISTER	

STOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
IMMEDIATE DATA															

SUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER			OPMODE			EFFECTIVE ADDRESS MODE				REGISTER	

SUBA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER			OPMODE			EFFECTIVE ADDRESS					
											MODE		REGISTER		

SUBI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	SIZE		EFFECTIVE ADDRESS MODE REGISTER					
WORD DATA (16 BITS)								BYTE DATA (8 BITS)							
LONG DATA (32 BITS)															

SUBQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	DATA			1	SIZE		EFFECTIVE ADDRESS					
										MODE		REGISTER			

SUBX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	REGISTER Dy/Ay				1	SIZE			0	0	R/M	REGISTER Dx/Ax

SWAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	REGISTER		

TAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

TBLS,TBLSN

Table Lookup and Interpolate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	REGISTER Dx			1	R	0	1	SIZE		0	0	0	0	0	0

Data Register Interpolate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	REGISTER Dym		
0	REGISTER Dx			1	R	0	0	SIZE		0	0	0	REGISTER Dyn		

TBLU,TBLUN

Table Lookup and Interpolate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	EFFECTIVE ADDRESS MODE REGISTER					
0	REGISTER Dx			0	R	0	1	SIZE		0	0	0	0	0	0

Data Register Interpolate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	REGISTER Dym		
0	REGISTER Dx			0	R	0	0	SIZE		0	0	0	REGISTER Dyn		

TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	VECTOR			

TRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	CONDITION				1	1	1	1	1	OPMODE		
OPTIONAL WORD															
OR LONG WORD															

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE		REGISTER			

UNLK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	REGISTER		

UNPK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	REGISTER Dy/Ay			1	1	0	0	0	R/M	REGISTER Dx/Ax		
16-BIT EXTENSION: ADJUSTMENT															

APPENDIX A

PROCESSOR INSTRUCTIONS SUMMARY

This appendix provides a quick reference of all the M68000 Family instructions listed by processors and the addressing modes used.

Table A-1 lists all the M68000 Family instructions by opcode and indicates which processors they apply to.

Table A-1. M68000 Family Instructions Processor Cross-Reference

Opcode	MC68000	MC68008	MC68010	MC68020	MC68030	MC68040	MC68881 MC68882	MC68851	CPU32
ABCD	X	X	X	X	X	X			X
ADD	X	X	X	X	X	X			X
ADDA	X	X	X	X	X	X			X
ADDI	X	X	X	X	X	X			X
ADDQ	X	X	X	X	X	X			X
ADDX	X	X	X	X	X	X			X
AND	X	X	X	X	X	X			X
ANDI	X	X	X	X	X	X			X
ANDI to CCR	X	X	X	X	X	X			X
ANDI to SR*	X	X	X	X	X	X			X
ASL, ASR	X	X	X	X	X	X			X
Bcc	X	X	X	X	X	X			X
BCHG	X	X	X	X	X	X			X
BCLR	X	X	X	X	X	X			X
BFCHG				X	X	X			
BFCLR				X	X	X			
BFEXTS				X	X	X			
BFEXTU				X	X	X			
BFFFO				X	X	X			
BFINS				X	X	X			
BFSET				X	X	X			
BFTST				X	X	X			
BGND									X

*PRIVILEGED (Supervisor) Instruction

**Table A-1. M68000 Family Instructions Processor
Cross-Reference (Continued)**

Opcode	MC68000	MC68008	MC68010	MC68020	MC68030	MC68040	MC68881 MC68882	MC68851	CPU32
BKPT			X	X	X	X			X
BRA	X	X	X	X	X	X			X
BSET	X	X	X	X	X	X			X
BSR	X	X	X	X	X	X			X
BTST	X	X	X	X	X	X			X
CALLM				X					
CAS, CAS2				X	X	X			
CHK	X	X	X	X	X	X			X
CHK2				X	X	X			X
CINV*						X			
CLR	X	X	X	X	X	X			X
CMP	X	X	X	X	X	X			X
CMPA	X	X	X	X	X	X			X
CMPI	X	X	X	X	X	X			X
CMPM	X	X	X	X	X	X			X
CMP2				X	X	X			X
cpBcc				X	X				
cpDBcc				X	X				
cpGEN				X	X				
cpRESTORE*				X	X				
cpSAVE*				X	X				
cpScc				X	X				
cpTRAPcc				X	X				
CPUSH*						X			
DBcc	X	X	X	X	X	X			X
DIVS	X	X	X	X	X	X			X
DIVSL				X	X	X			X
DIVU	X	X	X	X	X	X			X
DIVUL				X	X	X			X
EOR	X	X	X	X	X	X			X
EORI	X	X	X	X	X	X			X
EORI to CCR	X	X	X	X	X	X			X
EORI to SR*	X	X	X	X	X	X			X
EXG	X	X	X	X	X	X			X
EXT	X	X	X	X	X	X			X

*PRIVILEGED (Supervisor) Instruction

**Table A-1. M68000 Family Instructions Processor
Cross-Reference (Continued)**

Opcode	MC68000	MC68008	MC68010	MC68020	MC68030	MC68040	MC68881 MC68882	MC68851	CPU32
EXTB				X	X	X			X
FABS						X	X		
FSABS,FDABS						X			
FACOS						Note 1	X		
FADD						X	X		
FSADD, FDADD						X			
FASIN						Note 1	X		
FATAN						Note 1	X		
FATANH						Note 1	X		
FBcc						X	X		
FCMP						X	X		
FCOS						Note 1	X		
FCOSH						Note 1	X		
FDBcc						X	X		
FDIV						X	X		
FSDIV,FDDIV						X			
FETOX						Note 1	X		
FETOXM1						Note 1	X		
FGETEXP						Note 1	X		
FGETMAN						Note 1	X		
FINT						Note 1	X		
FINTRZ						Note 1	X		
FLOG10						Note 1	X		
FLOG2						Note 1	X		
FLOGN						Note 1	X		
FLOGNP1						Note 1	X		
FMOD						Note 1	X		
FMOVE						X	X		
FSMOVE,FDMOVE						X			
FMOVECR						Note 1	X		
FMOVEM						X	X		
FMUL						X	X		
FSMUL,FDMUL						X			
FNEG						X	X		
FSNEG, FDNEG						X			

NOTES

1. These instructions are software supported on the MC68040.

**Table A-1. M68000 Family Instructions Processor
Cross-Reference (Continued)**

Opcode	MC68000	MC68008	MC68010	MC68020	MC68030	MC68040	MC68881 MC68882	MC68851	CPU32
FNOP						X	X		
FREM						Note 1	X		
FRESTORE*						X	X		
FSAVE*						X	X		
FSCALE						Note 1	X		
FScc						X	X		
FSGLDIV						Note 1	X		
FSGLMUL						Note 1	X		
FSIN						Note 1	X		
FSINCOS						Note 1	X		
FSINH						Note 1	X		
FSQRT						X	X		
FSSQRT,FDSQRT						X			
FSUB						X	X		
FSSUB,FDSUB						X			
FTAN						Note 1	X		
FTANH						Note 1	X		
FTENTOX						Note 1	X		
FTRAPcc						X	X		
FTST						X	X		
FTWOTOX						Note 1	X		
ILLEGAL	X	X	X	X	X	X			X
JMP	X	X	X	X	X	X			X
JSR	X	X	X	X	X	X			X
LEA	X	X	X	X	X	X			X
LINK	X	X	X	X	X	X			X
LPSTOP									X
LSL, LSR	X	X	X	X	X	X			X
MOVE	X	X	X	X	X	X			X
MOVEA	X	X	X	X	X	X			X
MOVE from CCR			X	X	X	X			X
MOVE to CCR	X	X	X	X	X	X			X
MOVE from SR*	Note 2	Note 2	X	X	X	X			X

NOTES

1. These instructions are software supported on the MC68040.
 2. This instruction is not privileged for the MC68000 and MC68008.
- *PRIVILEGED (Supervisor) Instruction

**Table A-1. M68000 Family Instructions Processor
Cross-Reference (Continued)**

Opcode	MC68000	MC68008	MC68010	MC68020	MC68030	MC68040	MC68881 MC68882	MC68851	CPU32
MOVE to SR*	X	X	X	X	X	X			X
MOVE USP*	X	X	X	X	X	X			X
MOVEC*			X	X	X	X			X
MOVEM	X	X	X	X	X	X			X
MOVEP	X	X	X	X	X	X			X
MOVEQ	X	X	X	X	X	X			X
MOVES*			X	X	X	X			X
MOVE16						X			
MULS	X	X	X	X	X	X			X
MULU	X	X	X	X	X	X			X
NBCD	X	X	X	X	X	X			X
NEG	X	X	X	X	X	X			X
NEGX	X	X	X	X	X	X			X
NOP	X	X	X	X	X	X			X
NOT	X	X	X	X	X	X			X
OR	X	X	X	X	X	X			X
ORI	X	X	X	X	X	X			X
ORI to CCR	X	X	X	X	X	X			X
ORI to SR*	X	X	X	X	X	X			X
PACK				X	X	X			
PBcc*								X	
PDBcc*								X	
PEA	X	X	X	X	X	X			X
PFLUSH*					X	X		X	
PFLUSHA*					X			X	
PFLUSHS*								X	
PFLUSHR*								X	
PLOAD*					X			X	
PMOVE*					X			X	
PRESTORE*								X	
PSAVE*								X	
PScc*								X	
PTEST*					X	X		X	
PTRAPcc*								X	
PVALID								X	

*PRIVILEGED (Supervisor) Instruction



**Table A-1. M68000 Family Instructions Processor
Cross-Reference (Concluded)**

Opcode	MC68000	MC68008	MC68010	MC68020	MC68030	MC68040	MC68881 MC68882	MC68851	CPU32
RESET*	X	X	X	X	X	X			X
ROL, ROR	X	X	X	X	X	X			X
ROXL, ROXR	X	X	X	X	X	X			X
RTD			X	X	X	X			X
RTE*	X	X	X	X	X	X			X
RTM				X					
RTR	X	X	X	X	X	X			X
RTS	X	X	X	X	X	X			X
SBCD	X	X	X	X	X	X			X
Scc	X	X	X	X	X	X			X
STOP*	X	X	X	X	X	X			X
SUB	X	X	X	X	X	X			X
SUBA	X	X	X	X	X	X			X
SUBI	X	X	X	X	X	X			X
SUBQ	X	X	X	X	X	X			X
SUBX	X	X	X	X	X	X			X
SWAP	X	X	X	X	X	X			X
TAS	X	X	X	X	X	X			X
TBLS, TBLSN									X
TBLU, TBLUN									X
TRAP	X	X	X	X	X	X			X
TRAP _{cc}				X	X	X			X
TRAPV	X	X	X	X	X	X			X
TST	X	X	X	X	X	X			X
UNLK	X	X	X	X	X	X			X
UNPK				X	X	X			

NOTES

1. These instructions are software supported on the MC68040.
 2. This instruction is not privileged for the MC68000 and MC68008.
- *PRIVILEGED (Supervisor) Instruction

Table A-2 lists the M68000 Family instructions by mnemonics followed by the descriptive name.

Table A-2. M68000 Family Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ANDI to CCR	AND Immediate to Condition Code
ANDI to SR	AND Immediate to Status Register
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BGND	Enter Background Mode
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CALLM	CALL Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CINV	Invalidate Cache Entries
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
cpBcc	Branch on Coprocessor Condition
cpDBcc	Test Coprocessor Condition Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRAPcc	Trap on Coprocessor Condition
CPUSH	Push then Invalidate Cache Entries
DBcc	Test Condition, Decrement, and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EORI to CCR	Exclusive OR Immediate to Condition Code
EORI to SR	Exclusive OR Immediate to Status Register
EXG	Exchange Registers
EXT, EXTB	Sign Extend

**Table A-2. M68000 Family Instruction Set
(Continued)**

Mnemonic	Description
FABS	Floating-Point Absolute Value
FSFABS,FDFABS	Floating-Point Absolute Value (Single/Double Precision)
FACOS	Floating-Point Arc Cosine
FADD	Floating-Point Add
FSADD,FDADD	Floating-Point Add (Single/Double Precision)
FASIN	Floating-Point Arc Sine
FATAN	Floating-Point Arc Tangent
FATANH	Floating-Point Hyperbolic Arc Tangent
FBcc	Floating-Point Branch
FCMP	Floating-Point Compare
FCOS	Floating-Point Cosine
FCOSH	Floating-Point Hyperbolic Cosine
FDBcc	Floating-Point Decrement and Branch
FDIV	Floating-Point Divide
FSDIV,FDDIV	Floating-Point Divide (Single/Double Precision)
FETOX	Floating-Point e^x
FETOXM1	Floating-Point $e^x - 1$
FGETEXP	Floating-Point Get Exponent
FGETMAN	Floating-Point Get Mantissa
FINT	Floating-Point Integer Part
FINTRZ	Floating-Point Integer Part, Round-to-Zero
FLOG10	Floating-Point Log ₁₀
FLOG2	Floating-Point Log ₂
FLOGN	Floating-Point Log _e
FLOGNP1	Floating-Point Log _e (x + 1)
FMOD	Floating-Point Modulo Remainder
FMOVE	Move Floating-Point Register
FSMOVE,FDMOVE	Move Floating-Point Register (Single/Double Precision)
FMOVECR	Move Constant ROM
FMOVEM	Move Multiple Floating-Point Registers
FMUL	Floating-Point Multiply
FSMUL,FDMUL	Floating-Point Multiply (Single/Double Precision)
FNEG	Floating-Point Negate
FSNEG,FDNEG	Floating-Point Negate (Single/Double Precision)
FNOP	Floating-Point No Operation
FREM	IEEE Remainder
FRESTORE	Restore Floating-Point Internal State
FSAVE	Save Floating-Point Internal State
FSCALE	Floating-Point Scale Exponent
FScc	Floating-Point Set According to Condition
FSGLDIV	Single Precision Divide
FSGLMUL	Single Precision Multiply
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine
FSQRT	Floating-Point Square Root
FSSQRT,FDSQRT	Floating-Point Square Root (Single/Double Precision)
FSUB	Floating-Point Subtract
FSSUB,FDSUB	Floating-Point Subtract (Single/Double Precision)
FTAN	Tangent
FTANH	Hyperbolic Tangent
FTENTOX	Floating-Point 10^x
FTRAPcc	Floating-Point Trap-On Condition
FTST	Floating-Point Test
FTWOTOX	Floating-Point 2^x
ILLEGAL	Take Illegal Instruction Trap

**Table A-2. M68000 Family Instruction Set
(Continued)**

Mnemonic	Description
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LPSTOP	Low-Power Stop
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
MOVE to CCR	Move to Condition Code Register
MOVE from SR	Move from Status Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MOVE16	16-Byte Block Move
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
ORI to CCR	Inclusive OR Immediate to Condition Code
ORI to SR	Inclusive OR Immediate to Status Register
PACK	Pack BCD
PBcc	Branch on PMMU Condition
PDBcc	Test, Decrement, and Branch on PMMU Condition
PEA	Push Effective Address
PFLUSH	Flush Entry(ies) in the ATCs
PFLUSHA	Flush Entry(ies) in the ATCs
PFLUSHS	Flush Entry(ies) in the ATCs
PFLUSHR	Flush Entry(ies) in the ATCs and RPT Entries
PLOAD	Load an Entry into the ATC
PMOVE	Move PMMU Register
PRESTORE	PMMU Restore Function
PSAVE	PMMU Save Function
PScc	Set on PMMU Condition
PTEST	Test a Logical Address
PTRAPcc	Trap on PMMU Condition
PVALID	Validate a Pointer
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTM	Return from Module
RTR	Return and Restore
RTS	Return from Subroutine

**Table A-2. M68000 Family Instruction Set
(Concluded)**

Mnemonic	Description
SBCD Scc STOP SUB SUBA SUBI SUBQ SUBX SWAP	Subtract Decimal with Extend Set Conditionally Stop Subtract Subtract Address Subtract Immediate Subtract Quick Subtract with Extend Swap Register Words
TAS TBLS, TBLSN TBLU, TBLUN TRAP TRAPcc TRAPV TST	Test Operand and Set Signed Table Lookup with Interpolate Unsigned Table Lookup with Interpolate Trap Trap Conditionally Trap on Overflow Test Operand
UNLK UNPK	Unlink Unpack BCD

A.1 MC68000/MC68HC000/MC68008/MC68010 PROCESSORS

The following paragraphs provide information on the MC68000, MC68HC000, and MC68008 instruction set and addressing modes.

A.1.1 M68000, MC68HC000, MC68008, and MC68010 Instruction Set

Table A-3 lists the instructions used with the MC68000, MC68HC000, and MC68008 processors and Table A-4 lists the instructions used with MC68010.

Table A-3. MC68000, MC68HC000, and MC68008 Instruction Set

Mnemonic	Description
ABCD ADD ADDA ADDI ADDQ ADDX AND ANDI ANDI to CCR ANDI to SR ASL, ASR	Add Decimal with Extend Add Add Address Add Immediate Add Quick Add with Extend Logical AND Logical AND Immediate AND Immediate to Condition Code AND Immediate to Status Register Arithmetic Shift Left and Right
Bcc BCHG BCLR BRA BSET BSR BTST	Branch Conditionally Test Bit and Change Test Bit and Clear Branch Test Bit and Set Branch to Subroutine Test Bit
CHK CLR CMP CMPA CMPI CMPM	Check Register Against Bound Clear Compare Compare Address Compare Immediate Compare Memory to Memory
DBcc DIVS DIVU	Test Condition, Decrement, and Branch Signed Divide Unsigned Divide
EOR EORI EORI to CCR EORI to SR EXG EXT	Logical Exclusive OR Logical Exclusive OR Immediate Exclusive OR Immediate to Condition Code Exclusive OR Immediate to Status Register Exchange Registers Sign Extend
ILLEGAL	Take Illegal Instruction Trap



**Table A-3. MC68000, MC68HC000, and
MC68008 Instruction Set (Continued)**

Mnemonic	Description
JMP JSR	Jump Jump to Subroutine
LEA LINK LSL, LSR	Load Effective Address Link and Allocate Logical Shift Left and Right
MOVE MOVEA MOVE to CCR MOVE from SR MOVE to SR MOVE USP MOVEM MOVEP MOVEQ MULS MULU	Move Move Address Move to Condition Code Register Move from Status Register Move to Status Register Move User Stack Pointer Move Multiple Registers Move Peripheral Move Quick Signed Multiply Unsigned Multiply
NBCD NEG NEGX NOP NOT	Negate Decimal with Extend Negate Negate with Extend No Operation Logical Complement
OR ORI ORI to CCR ORI to SR	Logical Inclusive OR Logical Inclusive OR Immediate Inclusive OR Immediate to Condition Code Inclusive OR Immediate to Status Register
PEA	Push Effective Address
RESET ROL, ROR ROXL, ROXR RTE RTR RTS	Reset External Devices Rotate Left and Right Rotate with Extend Left and Right Return from Exception Return and Restore Return from Subroutine
SBCD SCC STOP SUB SUBA SUBI SUBQ SUBX SWAP	Subtract Decimal with Extend Set Conditionally Stop Subtract Subtract Address Subtract Immediate Subtract Quick Subtract with Extend Swap Register Words
TAS TRAP TRAPV TST	Test Operand and Set Trap Trap on Overflow Test Operand
UNLK	Unlink

Table A-4. MC68010 Instruction Set

Mnemonic	Description
ABCD ADD ADDA ADDI ADDQ ADDX AND ANDI ANDI to CCR ANDI to SR ASL, ASR	Add Decimal with Extend Add Add Address Add Immediate Add Quick Add with Extend Logical AND Logical AND Immediate AND Immediate to Condition Code AND Immediate to Status Register Arithmetic Shift Left and Right
Bcc BCHG BCLR BKPT BRA BSET BSR BTST	Branch Conditionally Test Bit and Change Test Bit and Clear Breakpoint Branch Test Bit and Set Branch to Subroutine Test Bit
CHK CLR CMP CMPA CMPI CMPM	Check Register Against Bound Clear Compare Compare Address Compare Immediate Compare Memory to Memory
DBcc DIVS DIVU	Test Condition, Decrement, and Branch Signed Divide Unsigned Divide
EOR EORI EORI to CCR EORI to SR EXG EXT	Logical Exclusive OR Logical Exclusive OR Immediate Exclusive OR Immediate to Condition Code Exclusive OR Immediate to Status Register Exchange Registers Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP JSR	Jump Jump to Subroutine
LEA LINK LSL, LSR	Load Effective Address Link and Allocate Logical Shift Left and Right

Table A-4. MC68010 Instruction Set (Continued)

Mnemonic	Description
MOVE MOVEA MOVE from CCR MOVE to CCR MOVE from SR MOVE to SR MOVE USP MOVEC MOVEM MOVEP MOVEQ MOVES MULS MULU	Move Move Address Move from Condition Code Register Move to Condition Code Register Move from Status Register Move to Status Register Move User Stack Pointer Move Control Register Move Multiple Registers Move Peripheral Move Quick Move Address Space Signed Multiply Unsigned Multiply
NBCD NEG NEGX NOP NOT	Negate Decimal with Extend Negate Negate with Extend No Operation Logical Complement
OR ORI ORI to CCR ORI to SR	Logical Inclusive OR Logical Inclusive OR Immediate Inclusive-OR Immediate to Condition Code Inclusive-OR Immediate to Status Register
PEA	Push Effective Address
RESET ROL, ROR ROXL, ROXR RTD RTE RTR RTS	Reset External Devices Rotate Left and Right Rotate with Extend Left and Right Return and Deallocate Return from Exception Return and Restore Return from Subroutine
SBCD Scc STOP SUB SUBA SUBI SUBQ SUBX SWAP	Subtract Decimal with Extend Set Conditionally Stop Subtract Subtract Address Subtract Immediate Subtract Quick Subtract with Extend Swap Register Words
TAS TRAP TRAPV TST	Test Operand and Set Trap Trap on Overflow Test Operand
UNLK	Unlink

A.1.2 MC68000, MC68HC000, MC68008, and MC68010 Addressing Modes

The MC68000, MC68HC000, MC68008, and MC68010 supports 14 addressing modes as shown in Table A-5.

**Table A-5. MC68000, MC68008, and MC68010
Data Addressing Modes**

Mode	Generation
Register Direct Addressing Data Register Direct Address Register Direct	EA = Dn EA = An
Absolute Data Addressing Absolute Short Absolute Long	EA = (Next Word) EA = (Next Two Words)
Program Counter Relative Addressing Relative with Offset Relative with Index and Offset	EA = (PC) + d ₁₆ EA = (PC) + d ₈
Register Indirect Addressing Register Indirect Postincrement Register Indirect Predecrement Register Indirect Register Indirect with Offset Indexed Register Indirect with Offset	EA = (An) EA = (An), An ♦ An + N An ♦ An - N, EA = (An) EA = (An) + d ₁₆ EA = (An) + (Xn) + d ₈
Immediate Data Addressing Immediate Quick Immediate	DATA = Next Word(s) Inherent Data
Implied Addressing Implied Register	EA = SR, USP, SSP, PC, VBR, SFC, DFC

NOTES:

EA = Effective Address

Dn = Data Register

An = Address Register

() = Contents of

PC = Program Counter

d₈ = 8-Bit Offset (Displacement)

d₁₆ = 16-Bit Offset (Displacement)

N = 1 for byte, 2 for word, and 4 for long word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundary.

♦ = Replaces

Xn = Address or Data Register used as Index Register

SR = Status Register

A.2 MC68020 PROCESSORS

The following paragraphs provide information on the MC68020 instruction set and addressing modes.

A.2.1 MC68020 Instruction Set

Table A-6 lists the instructions used with the MC68020 processors.

Table A-6. MC68020 Instruction Set Summary

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ANDI to CCR	AND Immediate to Condition Code
ANDI to SR	AND Immediate to Status Register
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit

Table A-6. MC68020 Instruction Set Summary (Continued)

Mnemonic	Description
CALLM CAS CAS2 CHK CHK2 CLR CMP CMPA CMPI CMPM CMP2 cpBcc cpDBcc cpGEN cpRESTORE cpSAVE cpScc cpTRAPcc	CALL Module Compare and Swap Operands Compare and Swap Dual Operands Check Register Against Bound Check Register Against Upper and Lower Bounds Clear Compare Compare Address Compare Immediate Compare Memory to Memory Compare Register Against Upper and Lower Bounds Branch on Coprocessor Condition Test Coprocessor Condition Decrement and Branch Coprocessor General Function Coprocessor Restore Function Coprocessor Save Function Set on Coprocessor Condition Trap on Coprocessor Condition
DBcc DIVS, DIVSL DIVU, DIVUL	Test Condition, Decrement, and Branch Signed Divide Unsigned Divide
EOR EORI EORI to CCR EORI to SR EXG EXT, EXTB	Logical Exclusive OR Logical Exclusive OR Immediate Exclusive OR Immediate to Condition Code Exclusive OR Immediate to Status Register Exchange Registers Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP JSR	Jump Jump to Subroutine
LEA LINK LSL, LSR	Load Effective Address Link and Allocate Logical Shift Left and Right
MOVE MOVEA MOVE from CCR MOVE to CCR MOVE from SR MOVE to SR MOVE USP MOVEC MOVEM MOVEP MOVEQ MOVES MULS MULU	Move Move Address Move from Condition Code Register Move to Condition Code Register Move from Status Register Move to Status Register Move User Stack Pointer Move Control Register Move Multiple Registers Move Peripheral Move Quick Move Alternate Address Space Signed Multiply Unsigned Multiply

Table A-6. MC68020 Instruction Set Summary (Concluded)

Mnemonic	Description
NBCD NEG NEGX NOP NOT	Negate Decimal with Extend Negate Negate with Extend No Operation Logical Complement
OR ORI ORI to CCR ORI to SR	Logical Inclusive OR Logical Inclusive OR Immediate Inclusive OR Immediate to Condition Code Inclusive OR Immediate to Status Register
PACK PEA	Pack BCD Push Effective Address
RESET ROL, ROR ROXL, ROXR RTD RTE RTM RTR RTS	Reset External Devices Rotate Left and Right Rotate with Extend Left and Right Return and Deallocate Return from Exception Return from Module Return and Restore Return from Subroutine
SBCD Scc STOP SUB SUBA SUBI SUBQ SUBX SWAP	Subtract Decimal with Extend Set Conditionally Stop Subtract Subtract Address Subtract Immediate Subtract Quick Subtract with Extend Swap Register Words
TAS TRAP TRAPcc TRAPV TST	Test Operand and Set Trap Trap Conditionally Trap on Overflow Test Operand
UNLK UNPK	Unlink Unpack BCD

A.2.2 MC68020 Addressing Modes

The MC68020 supports 18 addressing modes as shown in Table A-7.

Table A-7. MC68020 Data Addressing Modes

Addressing Modes	Syntax
Register Direct Data Register Direct Address Register Direct	Dn An
Register Indirect Address Register Indirect Address Register Indirect with Postincrement Address Register Indirect with Predecrement Address Register Indirect with Displacement	(An) (An) + – (An) (d ₁₆ ,An)
Register Indirect with Index Address Register Indirect with Index (8-Bit Displacement) Address Register Indirect with Index (Base Displacement)	(dg,An,Xn) (bd,An,Xn)
Memory Indirect Memory Indirect Postindexed Memory Indirect Preindexed	(([bd,An],Xn,od) ([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d ₁₆ ,PC)
Program Counter Indirect with Index PC Indirect with Index (8-Bit Displacement) PC Indirect with Index (Base Displacement)	(dg,PC,Xn) (bd,PC,Xn)
Program Counter Memory Indirect PC Memory Indirect Postindexed PC Memory Indirect Preindexed	(([bd,PC],Xn,od) ([bd,PC,Xn],od)
Absolute Absolute Short Absolute Long	(xxx).W (xxx).L
Immediate	#(data)

NOTES:

Dn = Data Register, D7–D0

An = Address Register, A7–A0

dg, d₁₆ = A two's-complement, or sign-extended displacement; added as part of the effective address calculation; size is 8 (dg) or 16 (d₁₆) bits; when omitted, assemblers use a value of zero.

Xn = Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.

bd = A two's-complement base displacement; when present, size can be 16 or 32 bits.

od = Outer displacement, added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.

PC = Program Counter

(data) = Immediate value of 8, 16, or 32 bits

() = Effective Address

[] = Use as indirect access to long-word address.

A.3 MC68030 PROCESSORS

The following paragraphs provide information on the MC68030 instruction set and addressing modes.

A.3.1 MC68030 Instruction Set

Table A-8 lists the instructions used with the MC68030 processors.

Table A-8. MC68030 Instruction Set Summary

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ANDI to CCR	AND Immediate to Condition Code
ANDI to SR	AND Immediate to Status Register
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit

Table A-8. MC68030 Instruction Set Summary (Continued)

Mnemonic	Description
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
cpBcc	Branch on Coprocessor Condition
cpDBcc	Test Coprocessor Condition Decrement and Branch
cpGEN	Coprocessor General Function
cpRESTORE	Coprocessor Restore Function
cpSAVE	Coprocessor Save Function
cpScc	Set on Coprocessor Condition
cpTRAPcc	Trap on Coprocessor Condition
DBcc	Test Condition, Decrement, and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EORI to CCR	Exclusive OR Immediate to Condition Code
EORI to SR	Exclusive OR Immediate to Status Register
EXG	Exchange Registers
EXT, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE from CCR	Move from Condition Code Register
MOVE to CCR	Move to Condition Code Register
MOVE from SR	Move from Status Register
MOVE to SR	Move to Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply

Table A-8. MC68030 Instruction Set Summary (Concluded)

Mnemonic	Description
NBCD NEG NEGX NOP NOT	Negate Decimal with Extend Negate Negate with Extend No Operation Logical Complement
OR ORI ORI to CCR ORI to SR	Logical Inclusive OR Logical Inclusive OR Immediate Inclusive OR Immediate to Condition Code Inclusive OR Immediate to Status Register
PACK PEA PFLUSH PFLUSHA PLOAD PMOVE PTEST	Pack BCD Push Effective Address Invalidate Entries in the ATC Invalidate all Entries in the ATC Load an Entry into the ATC Load an Entry into the ATC Get Information about Logical Address
RESET ROL, ROR ROXL, ROXR RTD RTE RTR RTS	Reset External Devices Rotate Left and Right Rotate with Extend Left and Right Return and Deallocate Return from Exception Return and Restore Return from Subroutine
SBCD Scc STOP SUB SUBA SUBI SUBQ SUBX SWAP	Subtract Decimal with Extend Set Conditionally Stop Subtract Subtract Address Subtract Immediate Subtract Quick Subtract with Extend Swap Register Words
TAS TRAP TRAPcc TRAPV TST	Test Operand and Set Trap Trap Conditionally Trap on Overflow Test Operand
UNLK UNPK	Unlink Unpack BCD

A.3.2 MC68030 Addressing Modes

The MC68030 supports 18 addressing modes as shown in Table A-9.

Table A-9. MC68030 Data Addressing Modes

Addressing Modes	Syntax
Register Direct Data Register Direct Address Register Direct	Dn An
Register Indirect Address Register Indirect Address Register Indirect with Postincrement Address Register Indirect with Predecrement Address Register Indirect with Displacement	(An) (An) + – (An) (d ₁₆ ,An)
Register Indirect with Index Address Register Indirect with Index (8-Bit Displacement) Address Register Indirect with Index (Base Displacement)	(d ₈ ,An,Xn) (bd,An,Xn)
Memory Indirect Memory Indirect Postindexed Memory Indirect Preindexed	([bd,An],Xn,od) ([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d ₁₆ ,PC)
Program Counter Indirect with Index PC Indirect with Index (8-Bit Displacement) PC Indirect with Index (Base Displacement)	(d ₈ ,PC,Xn) (bd,PC,Xn)
Program Counter Memory Indirect PC Memory Indirect Postindexed PC Memory Indirect Preindexed	([bd,PC],Xn,od) ([bd,PC,Xn],od)
Absolute Absolute Short Absolute Long	(xxx).W (xxx).L
Immediate	#(data)

NOTES:

- Dn = Data Register, D0–D7
- An = Address Register, A0–A7
- d₈, d₁₆ = A two's-complement, or sign-extended displacement; added as part of the effective address calculation; size is 8 (d₈) or 16 (d₁₆) bits; when omitted, assemblers use a value of zero.
- Xn = Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.
- bd = A two's-complement base displacement; when present, size can be 16 or 32 bits.
- od = Outer displacement, added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.
- PC = Program Counter
- (data) = Immediate value of 8, 16, or 32 bits
- () = Effective Address
- [] = Use as indirect access to long-word address.

A.4 MC68040 PROCESSOR

The following paragraphs provide information on the MC68040 instruction set and addressing modes.

A.4.1 MC68040 Instruction Set

Table A-10 lists the instructions used with the MC68040 processor.

Table A-10. MC68040 Instruction Set

Mnemonic	Description
ABCD ADD ADDA ADDI ADDQ ADDX AND ANDI ANDI to CCR ANDI to SR ASL, ASR	Add Decimal with Extend Add Add Address Add Immediate Add Quick Add with Extend Logical AND Logical AND Immediate AND Immediate to Condition Code AND Immediate to Status Register Arithmetic Shift Left and Right
Bcc BCHG BCLR BFCHG BFCLR BFEXTS BFEXTU BFFFO BFINS BFSET BFTST BKPT BRA BSET BSR BTST	Branch Conditionally Test Bit and Change Test Bit and Clear Test Bit Field and Change Test Bit Field and Clear Signed Bit Field Extract Unsigned Bit Field Extract Bit Field Find First One Bit Field Insert Test Bit Field and Set Test Bit Field Breakpoint Branch Test Bit and Set Branch to Subroutine Test Bit
CAS CAS2 CHK CHK2 CINV CLR CMP CMPA CMPI CMPM CMP2 CPUSH	Compare and Swap Operands Compare and Swap Dual Operands Check Register Against Bound Check Register Against Upper and Lower Bounds Invalidate Cache Entries Clear Compare Compare Address Compare Immediate Compare Memory to Memory Compare Register Against Upper and Lower Bounds Push then Invalidate Cache Entries
DBcc DIVS, DIVSL DIVU, DIVUL	Test Condition, Decrement, and Branch Signed Divide Unsigned Divide

Table A-10. MC68040 Instruction Set (Continued)

Mnemonic	Description
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EORI to CCR	Exclusive OR Immediate to Condition Code
EORI to SR	Exclusive OR Immediate to Status Register
EXG	Exchange Registers
EXT, EXTB	Sign Extend
FABS	Floating-Point Absolute Value
FSABS, FDABS	Floating-Point Absolute Value (Single/Double Precision)
FACOS*	Floating-Point Arc Cosine
FADD	Floating-Point Add
FSADD, FDADD	Floating-Point Add (Single/Double Precision)
FASIN*	Floating-Point Arc Sine
FATAN*	Floating-Point Arc Tangent
FATANH*	Floating-Point Hyperbolic Arc Tangent
FBcc	Floating-Point Branch
FCMP	Floating-Point Compare
FCOS*	Floating-Point Cosine
FCOSH*	Floating-Point Hyperbolic Cosine
FDBcc	Floating-Point Decrement and Branch
FDIV	Floating-Point Divide
FSDIV, FDDIV	Floating-Point Divide (Single/Double Precision)
FETOX*	Floating-Point e^x
FETOXM1*	Floating-Point $e^x - 1$
FGETEXP*	Floating-Point Get Exponent
FGETMAN*	Floating-Point Get Mantissa
FINT*	Floating-Point Integer Part
FINTRZ*	Floating-Point Integer Part, Round-to-Zero
FLOG10*	Floating-Point Log ₁₀
FLOG2*	Floating-Point Log ₂
FLOGN*	Floating-Point Log _e
FLOGNP1*	Floating-Point Log _e (x + 1)
FMOD*	Floating-Point Modulo Remainder
FMOVE	Move Floating-Point Register
FSMOVE, FDMOVE	Move Floating-Point Register (Single/Double Precision)
FMOVECR	Move Constant ROM
FMOVEM	Move Multiple Floating-Point Registers
FMUL	Floating-Point Multiply
FSMUL, FDMUL	Floating-Point Multiply (Single/Double Precision)
FNEG	Floating-Point Negate
FSNEG, FDNeg	Floating-Point Negate (Single/Double Precision)
FNOP	Floating-Point No Operation
FREM*	IEEE Remainder
FRESTORE	Restore Floating-Point Internal State
FSAVE	Save Floating-Point Internal State
FSCALE*	Floating-Point Scale Exponent
FScc	Floating-Point Set According to Condition
FSGLDIV*	Single Precision Divide
FSGLMUL*	Single Precision Multiply
FSIN*	Sine
FSINCOS*	Simultaneous Sine and Cosine
FSINH*	Hyperbolic Sine
FSQRT	Floating-Point Square Root
FSSQRT, FDSQRT	Floating-Point Square Root (Single/Double Precision)

Table A-10. MC68040 Instruction Set (Continued)

Mnemonic	Description
FSUB FSSUB, FDSUB FTAN* FTANH* FTENTOX* FTRAPcc FTST FTWOTOX*	Floating-Point Subtract Floating-Point Subtract (Single/Double Precision) Tangent Hyperbolic Tangent Floating-Point 10 ^x Floating-Point Trap-On Condition Floating-Point Test Floating-Point 2 ^x
ILLEGAL	Take Illegal Instruction Trap
JMP JSR	Jump Jump to Subroutine
LEA LINK LSL, LSR	Load Effective Address Link and Allocate Logical Shift Left and Right
MOVE MOVEA MOVE from CCR MOVE to CCR MOVE from SR MOVE to SR MOVE USP MOVEC MOVEM MOVEP MOVEQ MOVES MOVE16 MULS MULU	Move Move Address Move from Condition Code Register Move to Condition Code Register Move from Status Register Move to Status Register Move User Stack Pointer Move Control Register Move Multiple Registers Move Peripheral Move Quick Move Alternate Address Space 16-Byte Block Move Signed Multiply Unsigned Multiply
NBCD NEG NEGX NOP NOT	Negate Decimal with Extend Negate Negate with Extend No Operation Logical Complement
OR ORI ORI to CCR ORI to SR	Logical Inclusive OR Logical Inclusive OR Immediate Inclusive OR Immediate to Condition Code Inclusive OR Immediate to Status Register
PACK PEA PFLUSH PFLUSH A PTEST	Pack BCD Push Effective Address Flush Entry(ies) in the ATCs Flush all Entry(ies) in the ATCs Test a Logical Address

Table A-10. MC68040 Instruction Set (Concluded)

Mnemonic	Description
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test Operand and Set
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

*These instructions are software supported.

A.4.2 MC68040 Addressing Modes

The MC68040 supports 18 addressing modes as shown in Table A-11.

Table A-11. MC68040 Data Addressing Modes

Addressing Modes	Syntax
Register Direct Data Register Direct Address Register Direct	Dn An
Register Indirect Address Register Indirect Address Register Indirect with Postincrement Address Register Indirect with Predecrement Address Register Indirect with Displacement	(An) {An} + - {An} (d ₁₆ ,An)
Register Indirect with Index Address Register Indirect with Index (8-Bit Displacement) Address Register Indirect with Index (Base Displacement)	(dg,An,Xn) (bd,An,Xn)
Memory Indirect Memory Indirect Postindexed Memory Indirect Preindexed	{[bd,An],Xn,od} {[bd,An,Xn],od}
Program Counter Indirect with Displacement	(d ₁₆ ,PC)
Program Counter Indirect with Index PC Indirect with Index (8-Bit Displacement) PC Indirect with Index (Base Displacement)	(dg,PC,Xn) (bd,PC,Xn)
Program Counter Memory Indirect PC Memory Indirect Postindexed PC Memory Indirect Preindexed	{[bd,PC],Xn,od} {[bd,PC,Xn],od}
Absolute Absolute Short Absolute Long	xxx.W xxx.L
Immediate	#<data>

NOTES:

Dn = Data Register, D7–D0

An = Address Register, A7–A0

dg, d₁₆ = A two's-complement or sign-extended displacement; added as part of the effective address calculation; size is 8 (dg) or 16 (d₁₆) bits; when omitted, assemblers use a value of zero.

Xn = Address or data register used as an index register; form is Xn.SIZE/SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.

bd = A two's-complement base displacement; when present, size can be 16 or 32 bits.

od = Outer displacement, added as part of effective address calculation after any memory indirection; use is optional with size of 16 or 32 bits.

PC = Program Counter

<data> = Immediate value of 8, 16, or 32 bits.

{ } = Effective Address

[] = Used as indirect access to long-word address.

A.5 MC68881/MC68882 PROCESSORS

The following paragraphs provide information on the MC68881/MC68882 instruction set and addressing modes.

A.5.1 MC68881/MC68882 Instruction Set

Table A-12 lists the instructions used with the MC68881/MC68882 processors.

Table A-12. MC68881/MC68882 Instruction Set

Mnemonic	Description
FABS	Floating-Point Absolute Value
FACOS	Floating-Point Arc Cosine
FADD	Floating-Point Add
FASIN	Floating-Point Arc Sine
FATAN	Floating-Point Arc Tangent
FATANH	Floating-Point Hyperbolic Arc Tangent
FBcc	Floating-Point Branch
FCMP	Floating-Point Compare
FCOS	Floating-Point Cosine
FCOSH	Floating-Point Hyperbolic Cosine
FDBcc	Floating-Point Decrement and Branch
FDIV	Floating-Point Divide
FETOX	Floating-Point e^x
FETOXM1	Floating-Point $e^x - 1$
FGETEXP	Floating-Point Get Exponent
FGETMAN	Floating-Point Get Mantissa
FINT	Floating-Point Integer Part
FINTRZ	Floating-Point Integer Part, Round-to-Zero
FLOG10	Floating-Point \log_{10}
FLOG2	Floating-Point \log_2
FLOGN	Floating-Point \log_e
FLOGNP1	Floating-Point $\log_e (x + 1)$
FMOD	Floating-Point Modulo Remainder
FMOVE	Move Floating-Point Register
FMOVECR	Move Constant ROM
FMOVEM	Move Multiple Floating-Point Registers
FMUL	Floating-Point Multiply
FNEG	Floating-Point Negate
FNOP	Floating-Point No Operation
FREM	IEEE Remainder
FRESTORE	Restore Floating-Point Internal State
FSAVE	Save Floating-Point Internal State
FSCALE	Floating-Point Scale Exponent
FScc	Floating-Point Set According to Condition
FSGLDIV	Single Precision Divide
FSGLMUL	Single Precision Multiply
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine
FSQRT	Floating-Point Square Root
FSUB	Floating-Point Subtract
FTAN	Tangent
FTANH	Hyperbolic Tangent
FTENTOX	Floating-Point 10^x
FTRAPcc	Floating-Point Trap-On Condition
FTST	Floating-Point Test
FTWOTOX	Floating-Point 2^x

A.5.2 MC68881/MC68882 Addressing Modes

The MC68881/MC68882 does not perform address calculations. When the floating-point coprocessor instructs the processor to transfer an operand via the coprocessor interface, the processor performs the addressing mode calculation requested in the instruction.

A.6 MC68851 PROCESSORS

The following paragraphs provide information on the MC68851 instruction set and addressing modes.

A.6.1 MC68851 Instruction Set

Table A-13 lists the instructions used with the MC68851 processor.

Table A-13. MC68851 Instruction Set

Mnemonic	Description
PBcc	Branch on PMMU Condition
PDBcc	Test, Decrement, and Branch on PMMU Condition
PFLUSH	Flush Entry(ies) in the ATCs
PFLUSHA	Flush Entry(ies) in the ATCs
PFLUSHS	Flush Entry(ies) in the ATCs
PFLUSHR	Flush Entry(ies) in the ATCs and RPT Entries
PLOAD	Load an Entry into the ATC
PMOVE	Move PMMU Register
PRESTORE	PMMU Restore Function
PSAVE	PMMU Save Function
PScc	Set on PMMU Condition
PTEST	Test a Logical Address
PTRAPcc	Trap on PMMU Condition
PVALID	Validate a Pointer

A.6.2 MC68851 Addressing Modes

The MC68851 supports the same addressing modes as the MC68020.

Introduction **1**

Integer Instructions **2**

Floating-Point Instructions **3**

Supervisor (Privileged) Instructions **4**

CPU32 Instruction Summary **5**

Instruction Format Summary **6**

Processor Instruction Summary **A**

1 Introduction

2 Integer Instructions

3 Floating-Point Instructions

4 Supervisor (Privileged) Instructions

5 CPU32 Instruction Summary

6 Instruction Format Summary

A Processor Instruction Summary



MOTOROLA

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; P.O. Box 80300; Cheung Sha Wan Post Office; Kowloon Hong Kong.

JAPAN: Nippon Motorola Ltd.; 3-20-1 Minamiazabu, Minato-ku, Tokyo 106 Japan.

013 3289-6